# Power Platform World Tour

PowerBIUG PowerAppsUG FlowUG

# Power BI – Advanced Data Modelling

Augustin Dokoza Bukvic & Felix Möller

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Session Objectives & Agenda

By the end of this course, you will be able to use DAX to create calculations in a *Power BI Desktop* data model.  Specifically you will be able to:

- Understand basic concepts of Data Modeling
- Understand the consequences of data model design decisions
- Understand concepts of calculated columns and measures
- Gain familiarity with standard DAX patterns & CALCULATE
- Understand evaluation contexts and their impact on calculations
- Gain ability to parse data modeling formulas

# Who we are?

**Augustin Bukvic**
Senior Consultant Analytics

**Felix Möller**
Senior Azure Analytics Architect

7+ years experience in Microsoft BI

# Avanade Analytics

Databricks & Azure Cloud Native solution experts across Solution Architecture, Data Engineering, Advanced Analytics, and Analytics Experience

**3000+**

Man years of Data & Analytics experience

**1000+**

Azure cloud native systems designed and/or built

**4,000+**

Global Data Engineering, Data Scientists and AI Consultants

**100+**

Managed Services clients on Azure Native Solutions

**80**

Locations across 24 countries

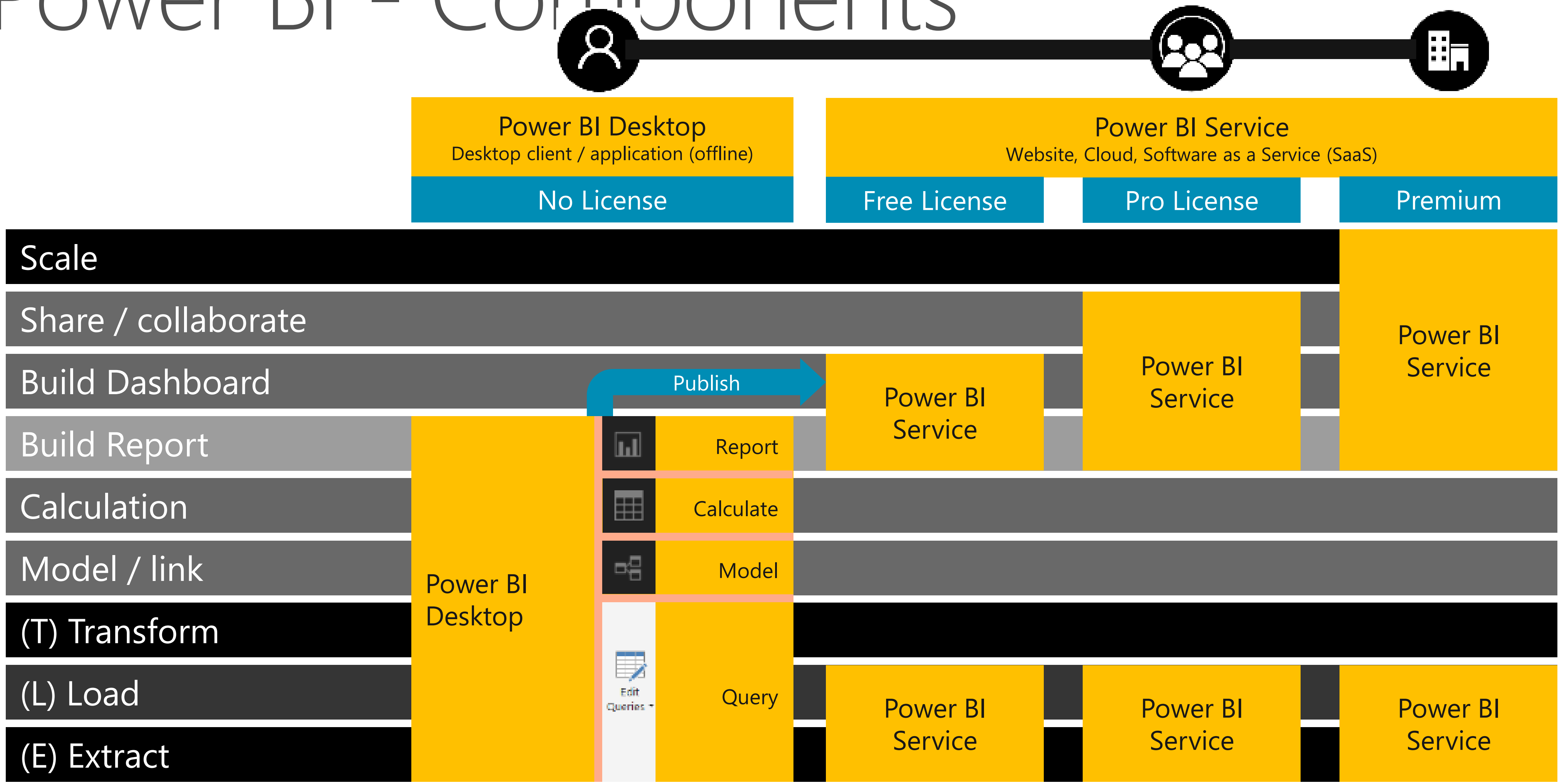
**46%**

of Global 500 companies as clients

# Who are you?

- What company do you work for?
- What experience  you have with Power BI?
- What are you source systems?
    - Enterprise Data Warehouse
    - Direct Access to Line of Business Systems (e.g. ERP)
- Do you use Analysis Services?
- What other Azure services are you using?

# Power BI - Components



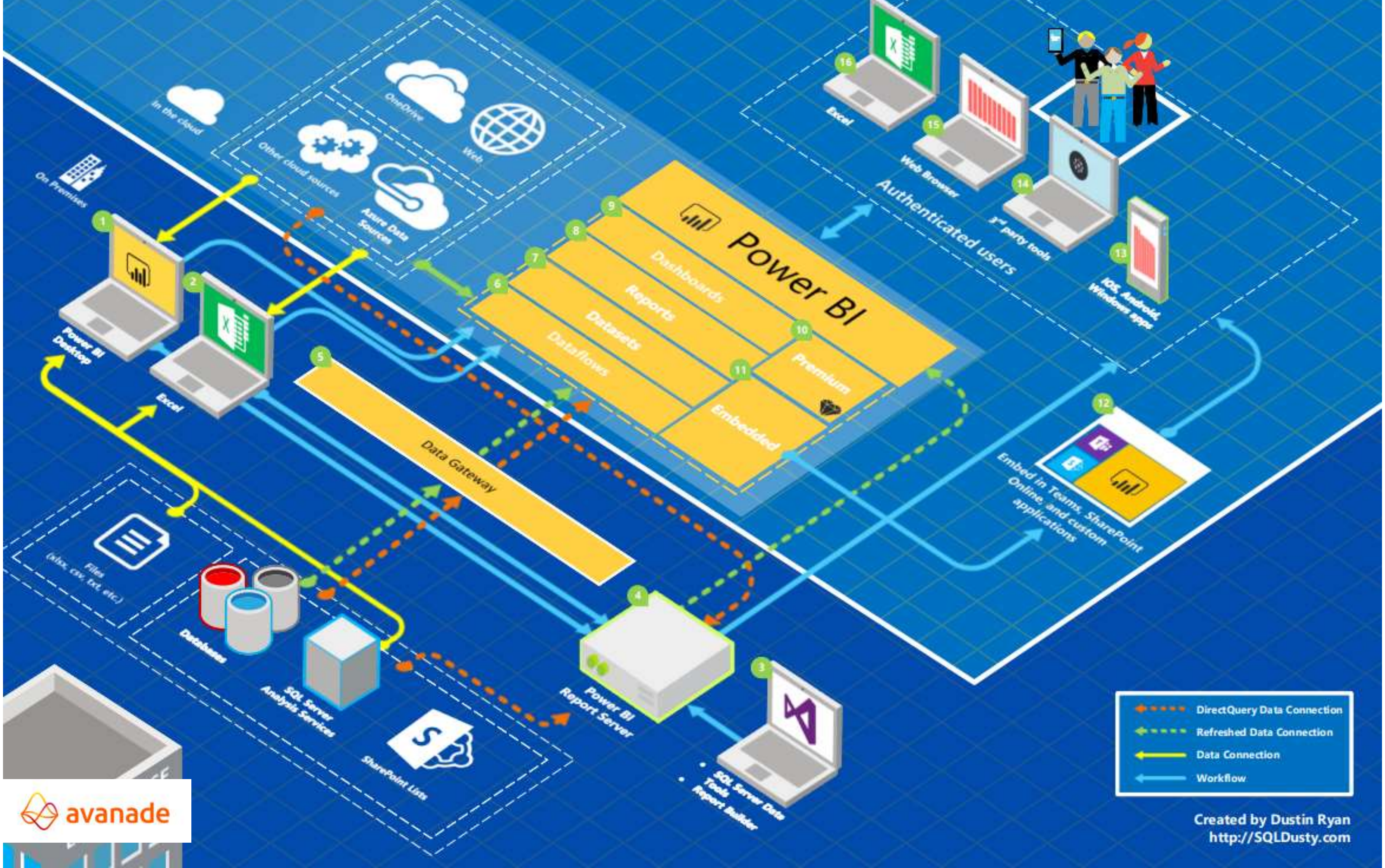
Power BI Desktop
Desktop client / application (offline)
No License

Power BI Service
Website, Cloud, Software as a Service (SaaS)
Free License
Pro License
Premium

Scale
Share / collaborate
Build Dashboard
Build Report
Calculation
Model / link
(T) Transform
(L) Load
(E) Extract

Power BI Desktop
Report
Calculate
Model
Edit Queries
Query
Publish

Power BI Service
Power BI Service
Power BI Service
Power BI Service
Power BI Service
Power BI Service



Created by Dustin Ryan
http://SQLDusty.com

# Housekeeping

- Keep an eye on the / breaks as scheduled
- Please confirm your attendance/sign the participant list
- Feel free to ask questions at any time (short QA at the end of each module)
- Enjoy learning
- Presentation and other materials can be dow

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

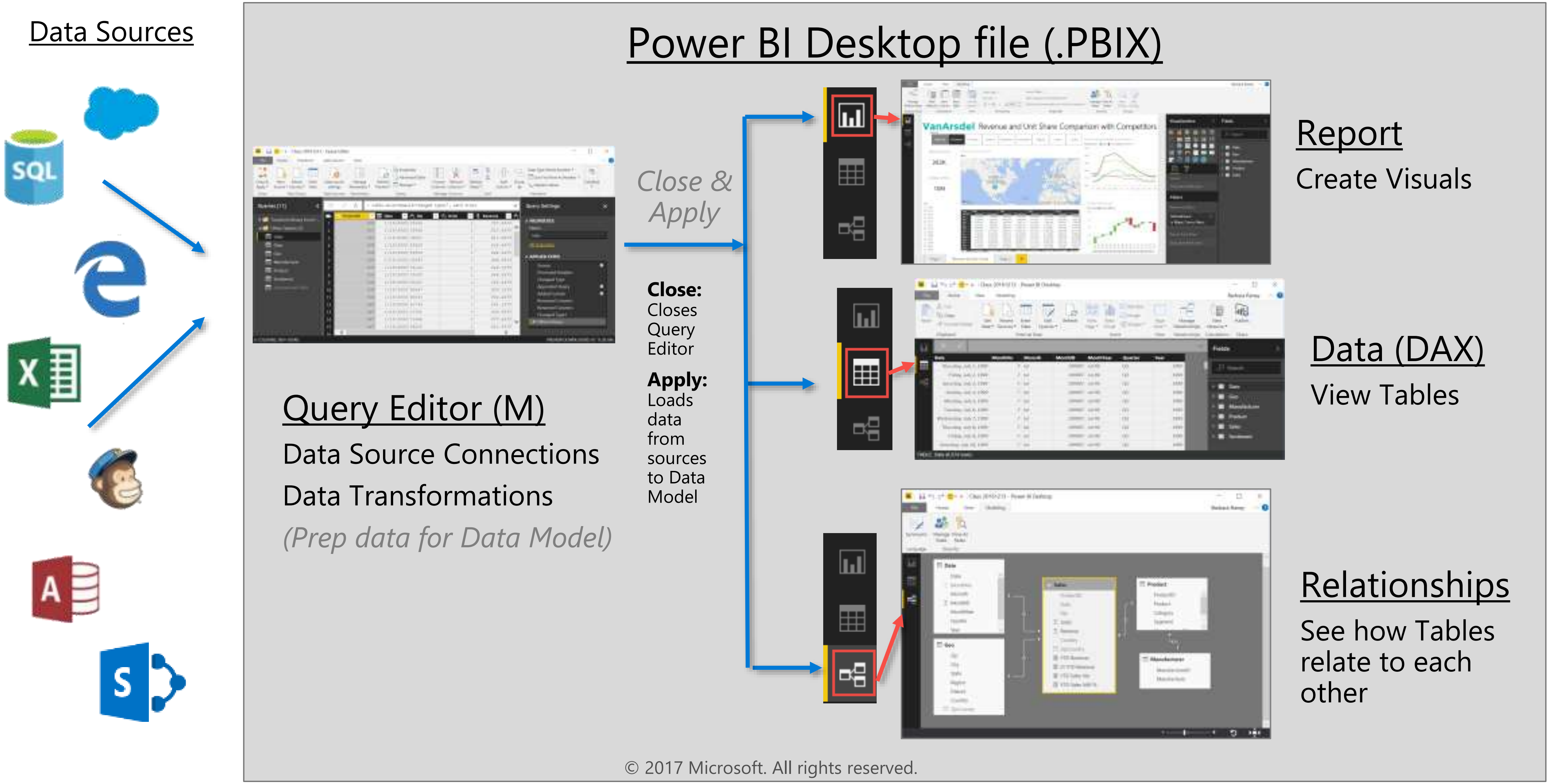
# Module
# Data Modeling Basics & *Power BI Desktop* Internals

# MODULE OBJECTIVES

- Understand what is meant by *data model* in the context of Power BI
- Understand the consequences of data model design decisions
- Understand Power BI's data storage architecture and use this knowledge to optimize performance
- Understand consequences of Power BI's data type handling

# Power BI Desktop Data Flow

Power BI



Data Sources

## Power BI Desktop file (.PBIX)

Query Editor (M)

Data Source Connections

Data Transformations

*(Prep data for Data Model)*

*Close & Apply*

**Close:** Closes Query Editor

**Apply:** Loads data from sources to Data Model

Report

Create Visuals

Data (DAX)

View Tables

Relationships

See how Tables relate to each other

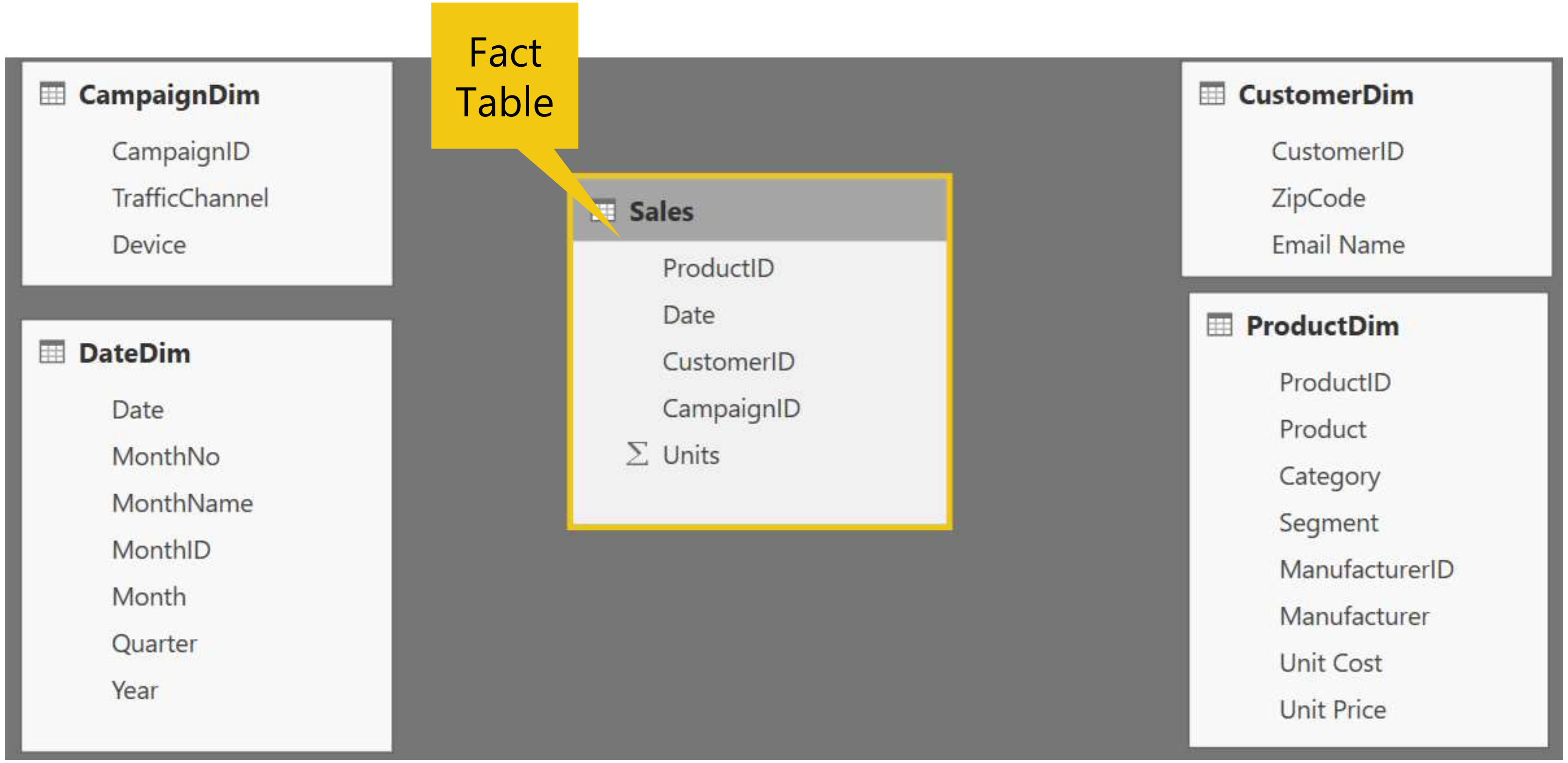
# What is a Data model?

A Power BI **Data Model** is a **collection of tables with relationships** which enable your business users to easily understand and explore their data to get business insights.

## Why is it important to have a Good Data model?

- Improves understandability of the data
- Increases performance of dependent processes and systems
- Increases resilience to change

# Components of a data model – Fact Table



Fact Table

**CampaignDim**
- CampaignID
- TrafficChannel
- Device

**DateDim**
- Date
- MonthNo
- MonthName
- MonthID
- Month
- Quarter
- Year

**Sales**
- ProductID
- Date
- CustomerID
- CampaignID
- Σ Units

**CustomerDim**
- CustomerID
- ZipCode
- Email Name

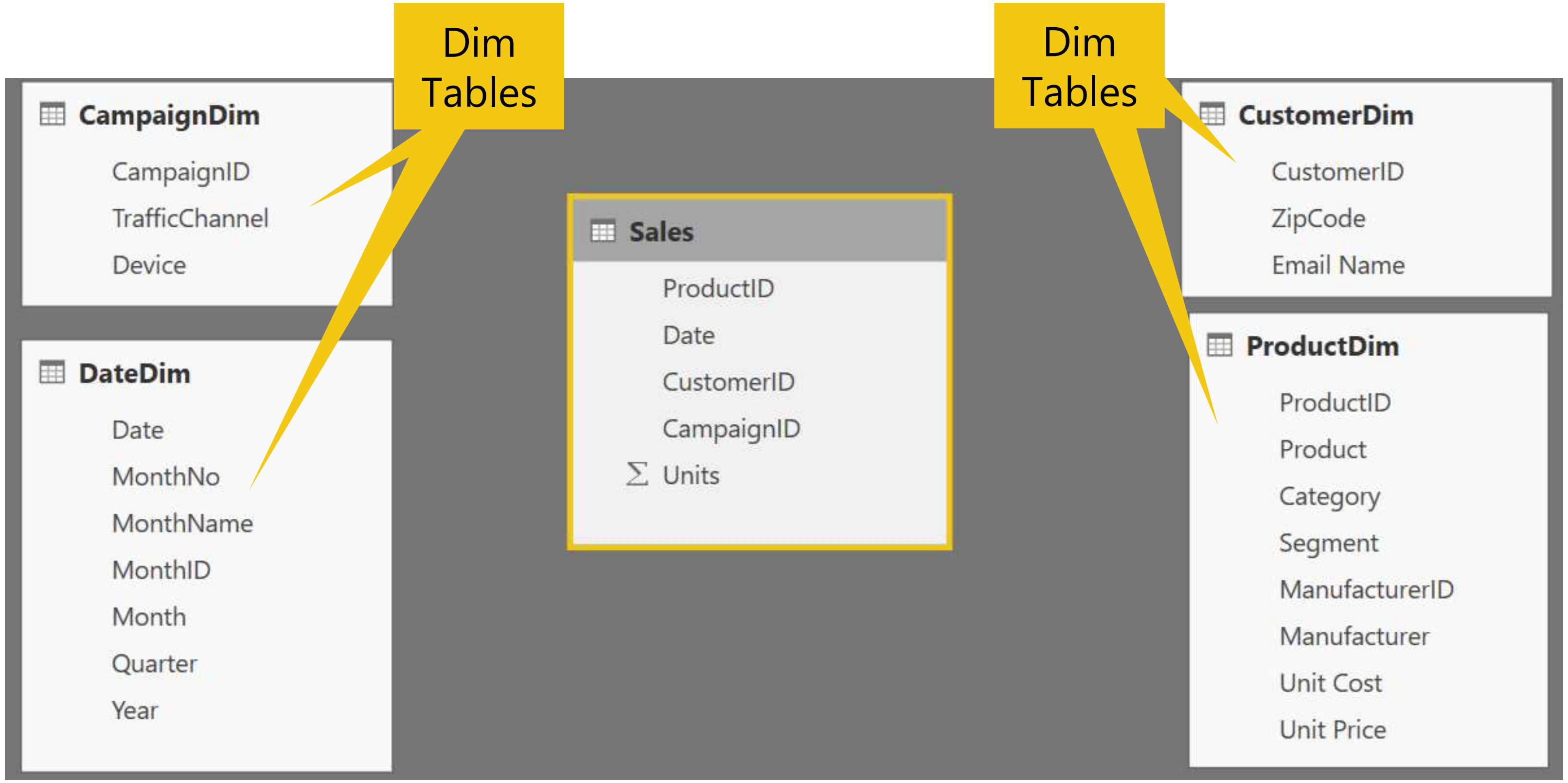
**ProductDim**
- ProductID
- Product
- Category
- Segment
- ManufacturerID
- Manufacturer
- Unit Cost
- Unit Price

## Fact Table

- Contains **Measures**
  (or items to be aggregated)
  of a business process
  **Examples:**
  - Transactions
  - Sales Revenue
  - Units
  - Cost
- Measures are usually sliceable.
  **Examples**: By Month,
  By Customer

# Components of a data model – Dim Table

Power BI



CampaignDim
CampaignID
TrafficChannel
Device

DateDim
Date
MonthNo
MonthName
MonthID
Month
Quarter
Year

Sales
ProductID
Date
CustomerID
CampaignID
Σ Units

CustomerDim
CustomerID
ZipCode
Email Name

ProductDim
ProductID
Product
Category
Segment
ManufacturerID
Manufacturer
Unit Cost
Unit Price

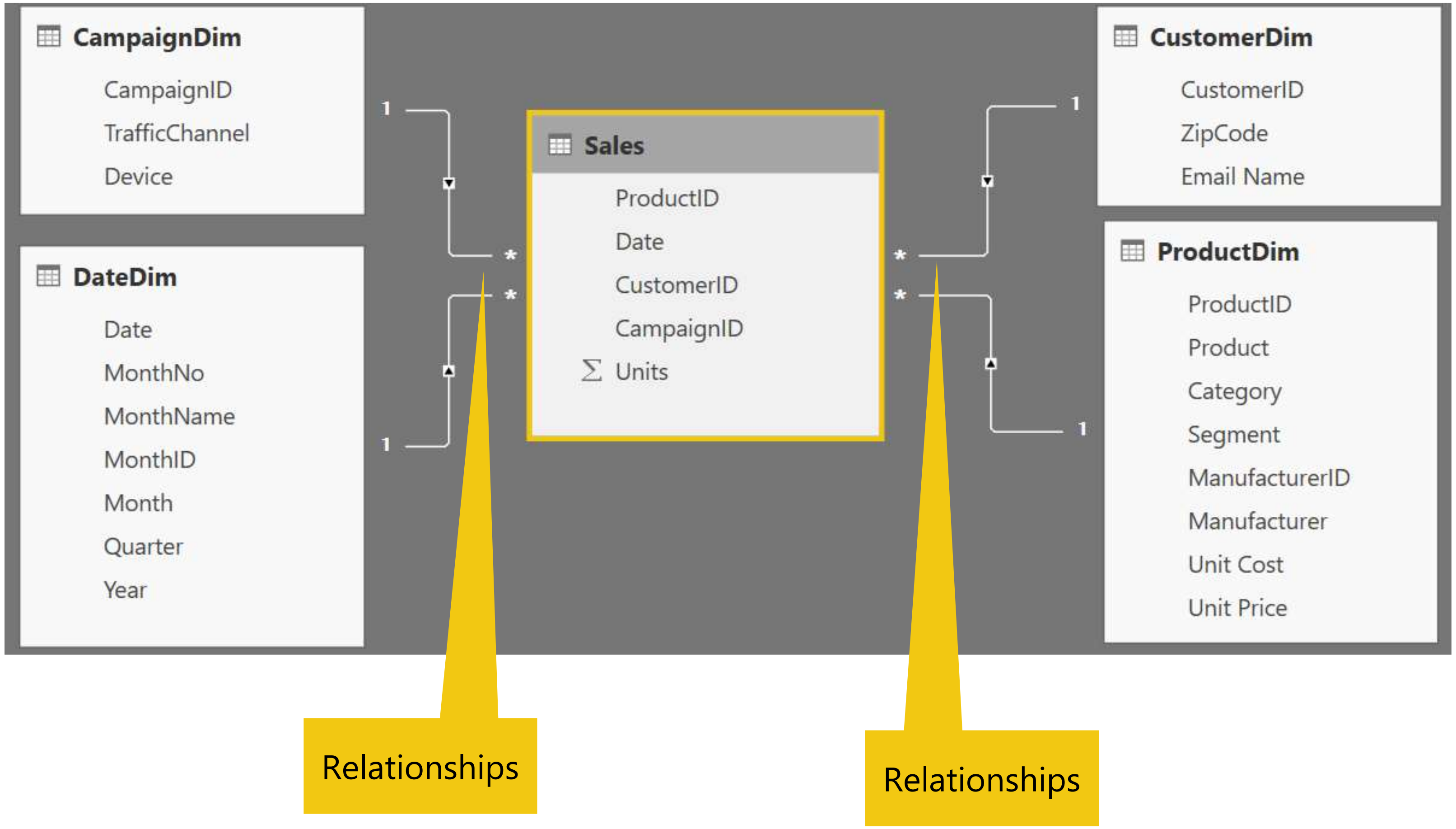
## Dim Table

A Dim (or Dimension) table contains descriptive attributes that define how a fact should roll up.

## Examples:

By month, By Customer, By Geo

# Components of a data model - Relationships



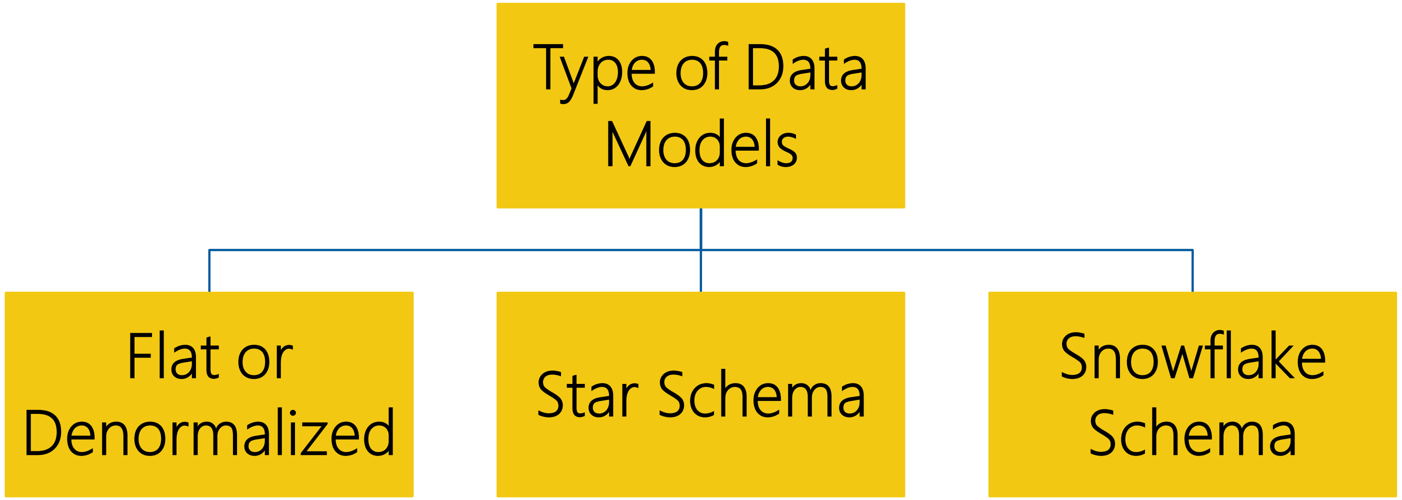
## Relationships

- Connection between a 2 tables (usually fact & Dim tables) using columns from each

- 3 kinds of Relationships
  - 1 to Many
  - 1 to 1
  - Many to Many

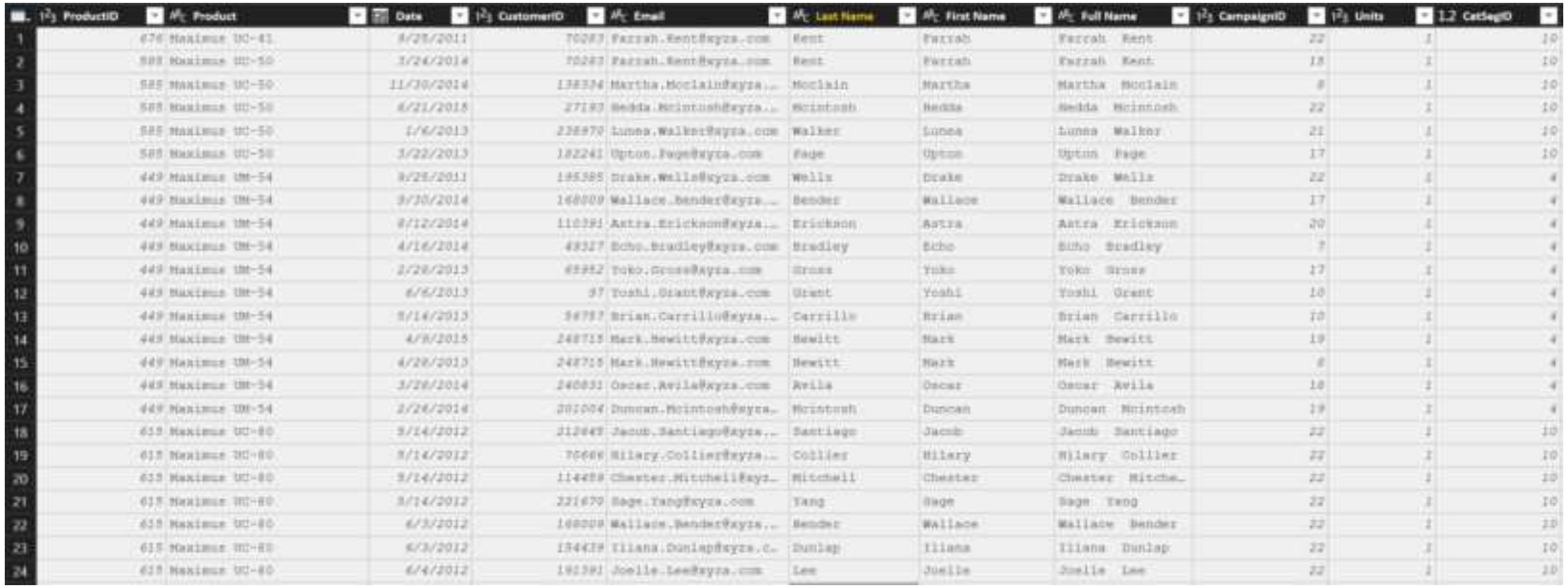
  (with a bridge table)

# Data Model Brings Facts and Dimensions Together



**Note:** This is not an exhaustive list, but are the most common model types used by Power BI.

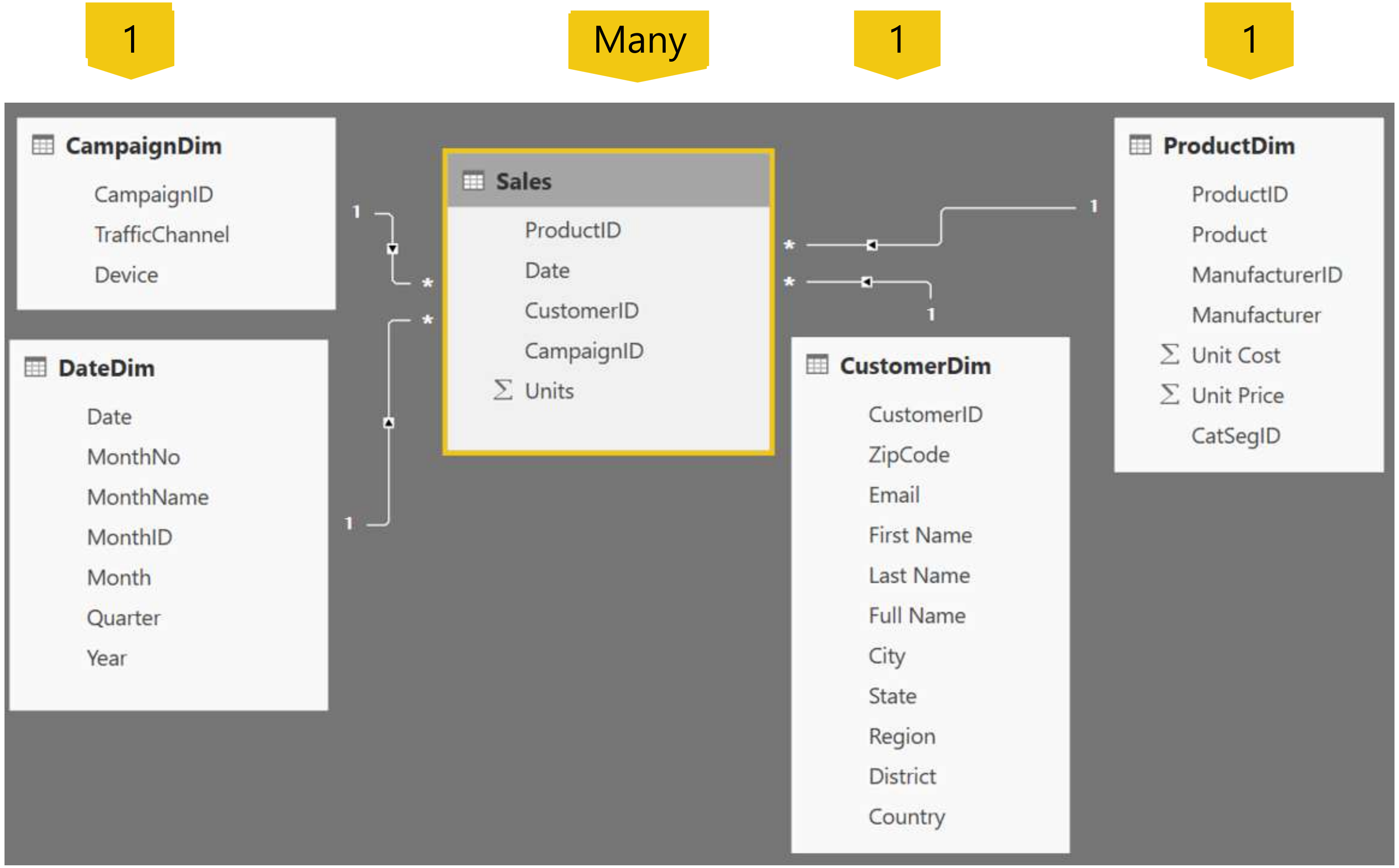
# Flat or Denormalized Schema



| | ProductID | Product | Date | CustomerID | Email | Last Name | First Name | Full Name | CampaignID | Units | CatSegID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 676 | Maximus UC-41 | 8/25/2011 | 70283 | Farrah.Kent@xyza.com | Kent | Farrah | Farrah Kent | 22 | 1 | 10 |
| 2 | 585 | Maximus UC-50 | 3/24/2014 | 70283 | Farrah.Kent@xyza.com | Kent | Farrah | Farrah Kent | 18 | 1 | 10 |
| 3 | 585 | Maximus UC-50 | 11/30/2014 | 138334 | Martha.Mcclain@xyza.… | Mcclain | Martha | Martha Mcclain | 8 | 1 | 10 |
| 4 | 585 | Maximus UC-50 | 6/21/2015 | 27193 | Hedda.Mcintosh@xyza.… | Mcintosh | Hedda | Hedda Mcintosh | 22 | 1 | 10 |
| 5 | 585 | Maximus UC-50 | 1/6/2013 | 238970 | Lunea.Walker@xyza.com | Walker | Lunea | Lunea Walker | 21 | 1 | 10 |
| 6 | 585 | Maximus UC-50 | 3/22/2013 | 182241 | Upton.Page@xyza.com | Page | Upton | Upton Page | 17 | 1 | 10 |
| 7 | 449 | Maximus UM-54 | 8/25/2011 | 195385 | Drake.Wells@xyza.com | Wells | Drake | Drake Wells | 22 | 1 | 4 |
| 8 | 449 | Maximus UM-54 | 9/30/2014 | 168009 | Wallace.Bender@xyza.… | Bender | Wallace | Wallace Bender | 17 | 1 | 4 |
| 9 | 449 | Maximus UM-54 | 8/12/2014 | 110381 | Astra.Erickson@xyza.… | Erickson | Astra | Astra Erickson | 20 | 1 | 4 |
| 10 | 449 | Maximus UM-54 | 4/16/2014 | 49327 | Echo.Bradley@xyza.com | Bradley | Echo | Echo Bradley | 7 | 1 | 4 |
| 11 | 449 | Maximus UM-54 | 2/28/2013 | 65882 | Yoko.Gross@xyza.com | Gross | Yoko | Yoko Gross | 17 | 1 | 4 |
| 12 | 449 | Maximus UM-54 | 6/6/2013 | 97 | Yoshi.Grant@xyza.com | Grant | Yoshi | Yoshi Grant | 10 | 1 | 4 |
| 13 | 449 | Maximus UM-54 | 5/14/2013 | 54757 | Brian.Carrillo@xyza.… | Carrillo | Brian | Brian Carrillo | 10 | 1 | 4 |
| 14 | 449 | Maximus UM-54 | 4/8/2015 | 248715 | Mark.Hewitt@xyza.com | Hewitt | Mark | Mark Hewitt | 19 | 1 | 4 |
| 15 | 449 | Maximus UM-54 | 4/28/2013 | 248715 | Mark.Hewitt@xyza.com | Hewitt | Mark | Mark Hewitt | 8 | 1 | 4 |
| 16 | 449 | Maximus UM-54 | 3/28/2014 | 240831 | Oscar.Avila@xyza.com | Avila | Oscar | Oscar Avila | 18 | 1 | 4 |
| 17 | 449 | Maximus UM-54 | 2/26/2014 | 201004 | Duncan.Mcintosh@xyza.… | Mcintosh | Duncan | Duncan Mcintosh | 19 | 1 | 4 |
| 18 | 615 | Maximus UC-80 | 5/14/2012 | 212845 | Jacob.Santiago@xyza.… | Santiago | Jacob | Jacob Santiago | 22 | 1 | 10 |
| 19 | 615 | Maximus UC-80 | 5/14/2012 | 70666 | Hilary.Collier@xyza.… | Collier | Hilary | Hilary Collier | 22 | 1 | 10 |
| 20 | 615 | Maximus UC-80 | 5/14/2012 | 114458 | Chester.Mitchell@xyz… | Mitchell | Chester | Chester Mitche… | 22 | 1 | 10 |
| 21 | 615 | Maximus UC-80 | 5/14/2012 | 221670 | Sage.Yang@xyza.com | Yang | Sage | Sage Yang | 22 | 1 | 10 |
| 22 | 615 | Maximus UC-80 | 6/3/2012 | 168009 | Wallace.Bender@xyza.… | Bender | Wallace | Wallace Bender | 22 | 1 | 10 |
| 23 | 615 | Maximus UC-80 | 6/3/2012 | 154439 | Iliana.Dunlap@xyza.c… | Dunlap | Iliana | Iliana Dunlap | 22 | 1 | 10 |
| 24 | 615 | Maximus UC-80 | 6/4/2012 | 191381 | Joelle.Lee@xyza.com | Lee | Joelle | Joelle Lee | 22 | 1 | 10 |

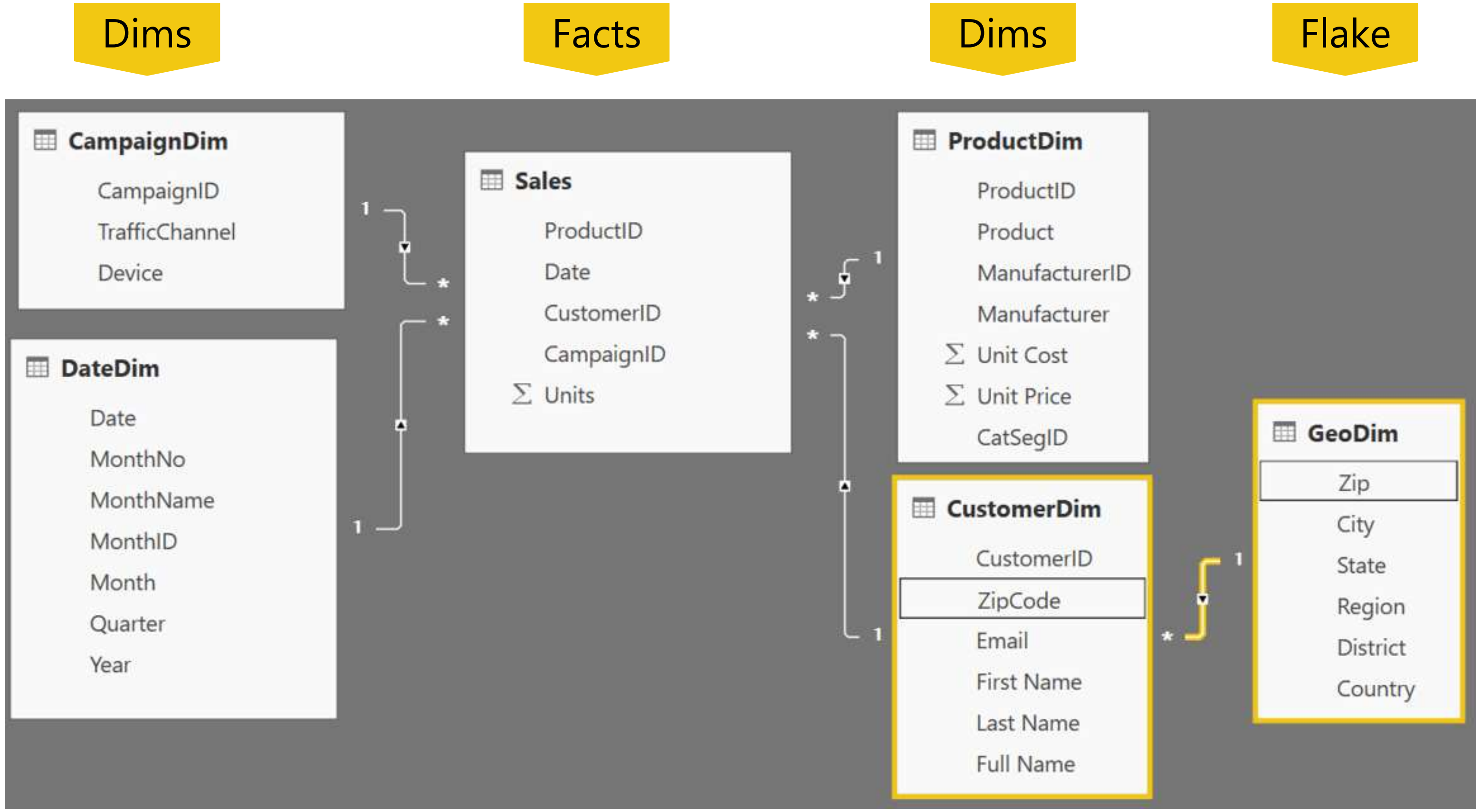
- All attributes for model exist in a **single** table
- **Highly inefficient**
- Model has extra copies of data > slow performance
- Size of a flat table can blow up really quickly as data model becomes complex

# Star Schema



- **Fact** table in the middle
- Surrounded by **Dims**
- Looks like a 'Star'
- Fact table is the "Many" side of the (one to many) relationship

# Snowflake Schema



**CampaignDim**
- CampaignID
- TrafficChannel
- Device

**DateDim**
- Date
- MonthNo
- MonthName
- MonthID
- Month
- Quarter
- Year

**Sales**
- ProductID
- Date
- CustomerID
- CampaignID
- Σ Units

**ProductDim**
- ProductID
- Product
- ManufacturerID
- Manufacturer
- Σ Unit Cost
- Σ Unit Price
- CatSegID

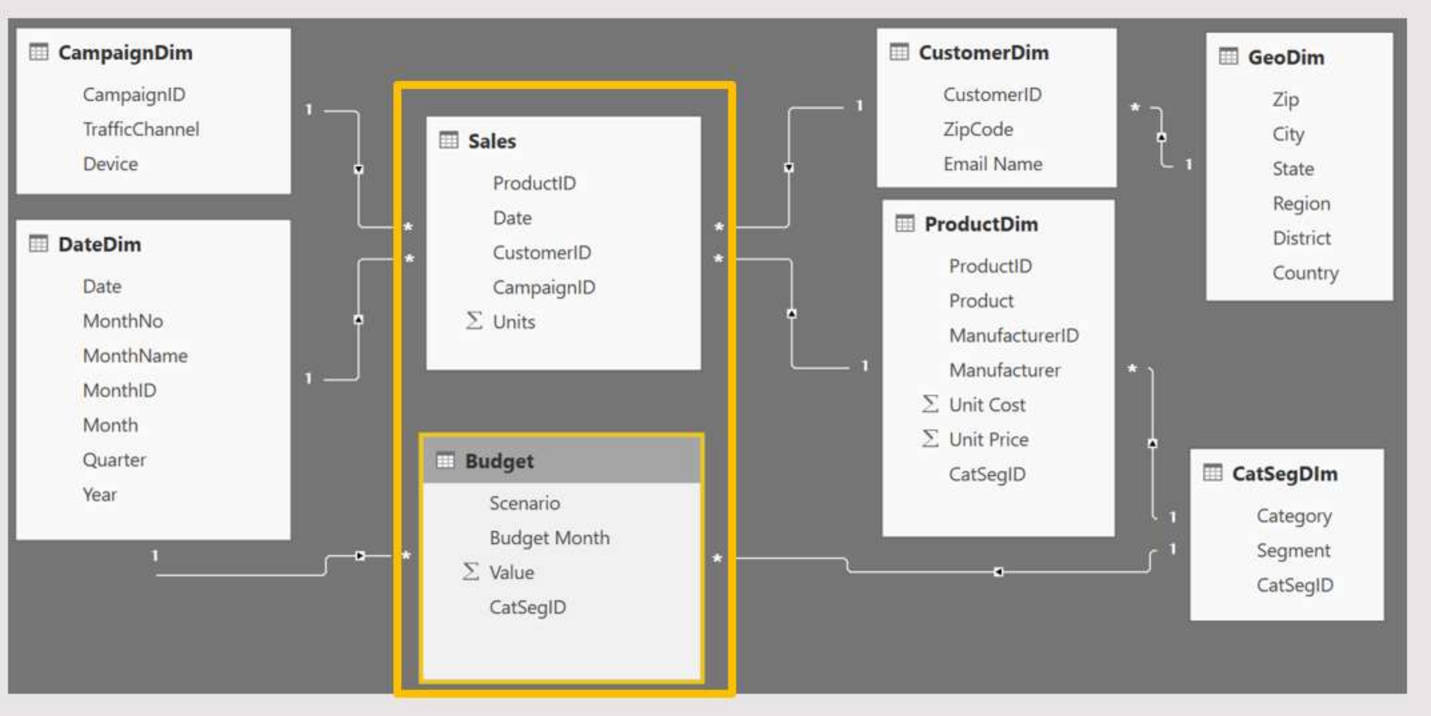
**CustomerDim**
- CustomerID
- ZipCode
- Email
- First Name
- Last Name
- Full Name

**GeoDim**
- Zip
- City
- State
- Region
- District
- Country

- Center is a Star schema
- Fact table in middle
- Surrounded by Dims
- Dims “snowflake” off of other Dims
- If you have many, it looks like a ‘Snowflake’
- Dim or Fact tables can be the “Many” side of the relationship

# Granularity & Multiple Fact Tables

Power BI



Sales (Daily by Product)

Budget (Monthly by Product Category & Product Segment)

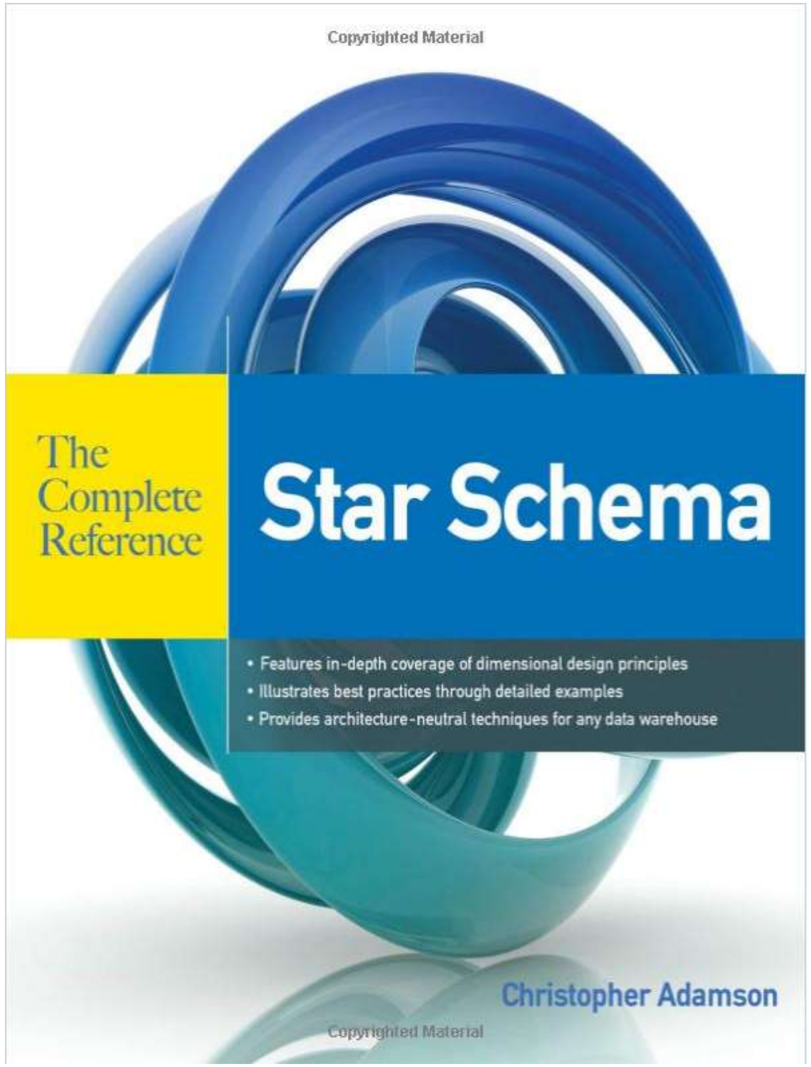
- Grain (**granularity)** measures the level of detail in a table

  Example:
  One row per **order or per Item**
  **Daily or Monthly date grain**

- If your facts have **very different granularities**, split them into **Multiple Fact** tables & connect them to shared dimensions at the lowest common granularity.

# Star Schema Book

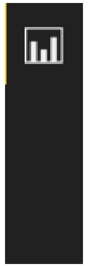
- Easy to understand
- Lots of inspiration for data models
- 486 pages
- https://www.amazon.de/dp/0071744320

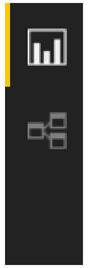

Copyrighted Material
The Complete Reference
Star Schema
• Features in-depth coverage of dimensional design principles
• Illustrates best practices through detailed examples
• Provides architecture-neutral techniques for any data warehouse
Christopher Adamson
Copyrighted Material

# Connection Types in Power BI

## How can I tell what Connection Type I have?

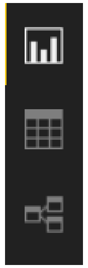
- Live Connect to SQL Analysis Services (SSAS) tabular
  - Report view only available





- DirectQuery to SQL or other relational source
  - Report & Relationship views available

- Import data into Power BI (creates a copy of the data)
  - Report, Data and Relationship views available



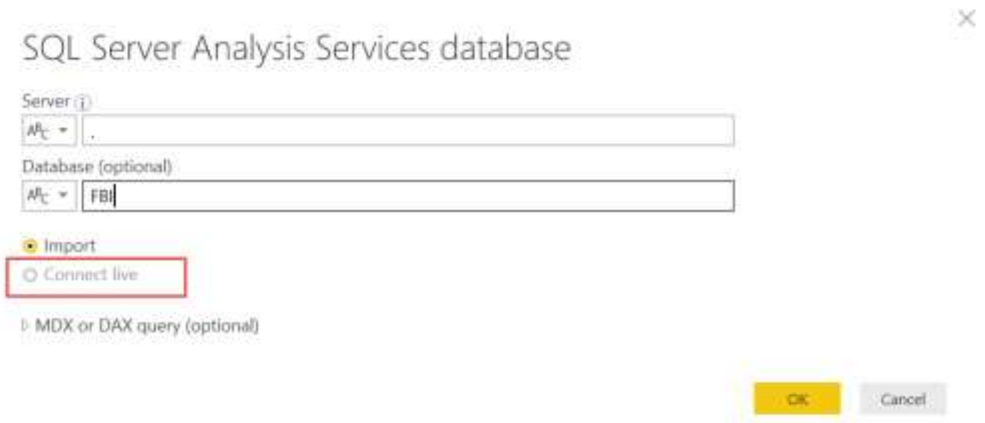
# Connection: Live Connect

- Live Connect to Multidimensional or Tabular
  - On Premise or Azure

- Only a single connection will be made and all modeling is done in the cube

- You can not add relationships or additional data source

- If allowed, you can add DAX measures



SQL Server Analysis Services database

Server: .

Database (optional): FBI

● Import

○ Connect live

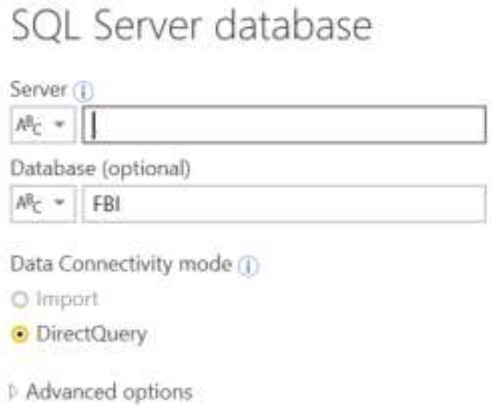
MDX or DAX query (optional)

# Connection: DirectQuery to Relational Source

- Direct Query to SQL or other relational source
  - On Premise or Azure

- All Data Sources are required to be DirectQuery, you cannot “Mashup” with Import sources or you get this message

- You can add relationships and DAX

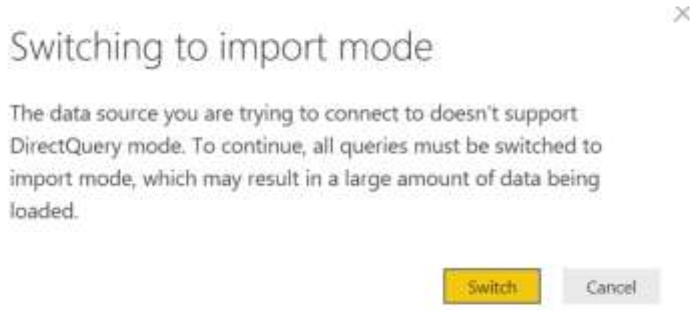


SQL Server database

Server

Database (optional)
FBI

Data Connectivity mode
○ Import
● DirectQuery



Switching to import mode

The data source you are trying to connect to doesn't support DirectQuery mode. To continue, all queries must be switched to import mode, which may result in a large amount of data being loaded.

# Connection: Import Mode

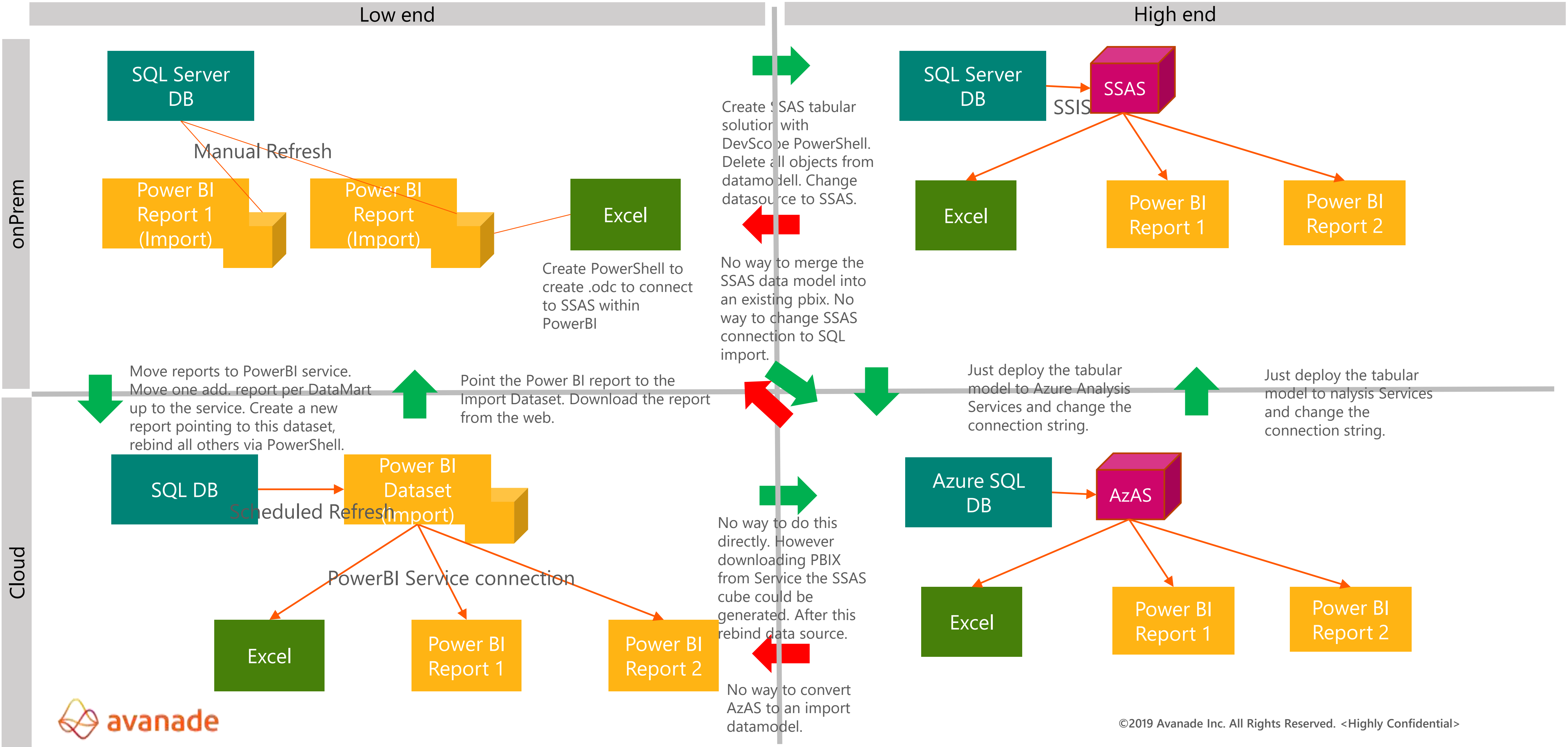
Power BI

## What is unique about Power BI Desktop in Import Mode?

- Columnar database
- In-memory database

**Let us understand some of the internals of Power BI Desktop !!**

# Import Mode cannot be changed easily



Low end

High end

onPrem

Cloud

SQL Server DB

Manual Refresh

Power BI Report 1 (Import)

Power BI Report (Import)

Excel

Create PowerShell to create .odc to connect to SSAS within PowerBI

Create SSAS tabular solution with DevScope PowerShell. Delete all objects from datamodell. Change datasource to SSAS.

No way to merge the SSAS data model into an existing pbix. No way to change SSAS connection to SQL import.

SQL Server DB

SSIS

SSAS

Excel

Power BI Report 1

Power BI Report 2

Move reports to PowerBI service. Move one add. report per DataMart up to the service. Create a new report pointing to this dataset, rebind all others via PowerShell.

Point the Power BI report to the Import Dataset. Download the report from the web.

Just deploy the tabular model to Azure Analysis Services and change the connection string.

Just deploy the tabular model to nalysis Services and change the connection string.

SQL DB

Scheduled Refresh

Power BI Dataset (Import)

PowerBI Service connection

Excel

Power BI Report 1

Power BI Report 2

No way to do this directly. However downloading PBIX from Service the SSAS cube could be generated. After this rebind data source.

No way to convert AzAS to an import datamodel.

Azure SQL DB

AzAS

Excel

Power BI Report 1

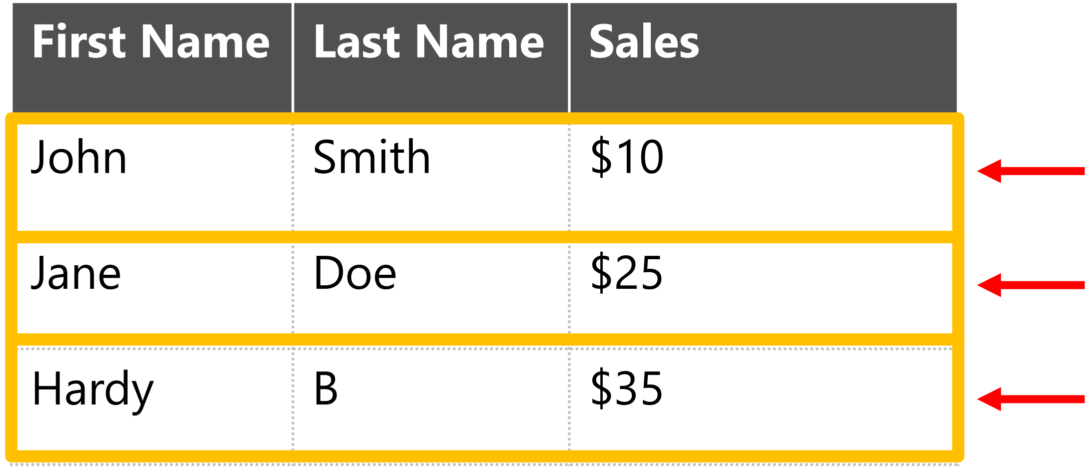
Power BI Report 2

avanade

# Columnar Database

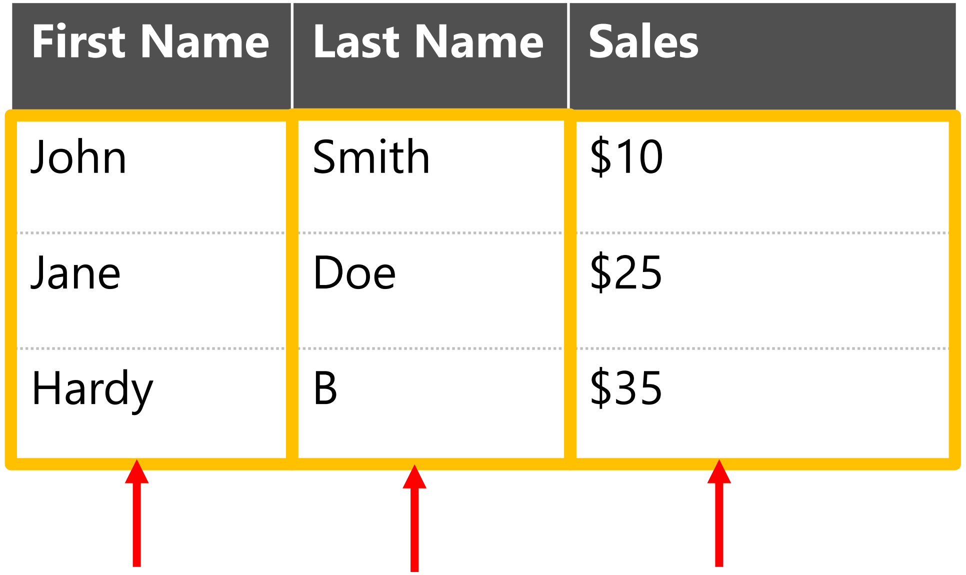
Power BI

## Row Based Database



| First Name | Last Name | Sales |
|---|---|---|
| John | Smith | $10 |
| Jane | Doe | $25 |
| Hardy | B | $35 |

## PBI - Columnar Database



| First Name | Last Name | Sales |
|---|---|---|
| John | Smith | $10 |
| Jane | Doe | $25 |
| Hardy | B | $35 |

- Stores **each row separately** (like a separate file)
- Retrieving multiple columns from a single row is fast
- Retrieving multiple rows from a single column is slower

- Stores **each column separately** (like a separate file)
- Retrieving multiple columns from a single row is slow
- Retrieving multiple rows from a single column is faster
- **Columnar databases are well suited for analytics**

# In-Memory Database

## PBI – In-Memory Database

- Data stored in **RAM (in memory)**
- RAM is all electronic – **Read/Write is fast**
- Laptops have smaller **RAM space (~8GB)**

Power BI compresses data to conserve space in RAM

# Compressing Data – Dictionary Encoding

## How Power BI Compresses Data – Dictionary Encoding

| Sale Id | Color | Sales Amount |
|---|---|---|
| 390a30e0-dc37 | Red | $10 |
| 390a30e1-dc37 | Green | $25 |
| 390a30e2-dc37 | Red | $35 |
| 390a30e3-dc37 | Red | $15 |
| 390a30e4-dc37 | Red | $25 |
| 390a30e5-dc37 | Green | $30 |
| 390a30e6-dc37 | Blue | $10 |
| 390a30e7-dc37 | Blue | $12 |
| 390a30e8-dc37 | Blue | $15 |
| 390a57f0-dc37 | Blue | $18 |
| 390a57f1-dc37 | Green | $25 |

**Red = 1** **Green = 2** **Blue = 3**

- Create a Dictionary to create an integer value for text string
- Storing 1,2,3 instead of “Red”, “Green”, “Blue” saves memory
- **Dictionary encoding is powerful when there are few unique values** in a column
  - Ex. Color column – Good for dictionary encoding
  - Ex. Sale ID – Bad for dictionary encoding

# Compressing Data – Run Length Encoding

## How Power BI Compresses Data – Run Length Encoding

| Sale Id | Color | Sales Amount | |
|---|---|---|---|
| 390a30e0-dc37 | Red | $10 | 1 |
| 390a30e1-dc37 | Green | $25 | 2 |
| 390a30e2-dc37 | Red | $35 | 1 |
| 390a30e3-dc37 | Red | $15 | 1 |
| 390a30e4-dc37 | Red | $25 | 1 |
| 390a30e5-dc37 | Green | $30 | 2 |
| 390a30e6-dc37 | Blue | $10 | 3 |
| 390a30e7-dc37 | Blue | $12 | 3 |
| 390a30e8-dc37 | Blue | $15 | 3 |
| 390a57f0-dc37 | Blue | $18 | 3 |
| 390a57f1-dc37 | Green | $25 | 2 |

### Run Length Encoding in Power BI

Where **Red** = **1** **Green** = **2** **Blue** = **3**

- Instead of storing - 1, 2, 1, 1, 1, 2, 3, 3, 3, 3, 2
- It Stores:

| | |
|---|---|
| 1 – 1 | (1 instance of One) |
| 1 – 2 | (1 instance of Two) |
| 3 – 1 | (3 instances of One) |
| 1 – 2 | (1 instance of Two) |
| 4 – 3 | (4 instances of Three) |
| 1 - 2 | (1 instance of Two) |

- **Run length encoding** is **very powerful when data is sorted** well and has few unique values

# Compression

Power BI

## Practical Example of Compression

Dashboard in a Day Class Data

| | |
|---|---:|
| Sales Fact | 420.0 MB |
| Dimensions | 4.4 MB |
| Int'l Sales | 32.4 MB |
| **Total Data** | **456.8 MB** |

Queries ONLY – No Data Loaded

| | |
|---|---:|
| **Query Metadata** | **113 KB** |

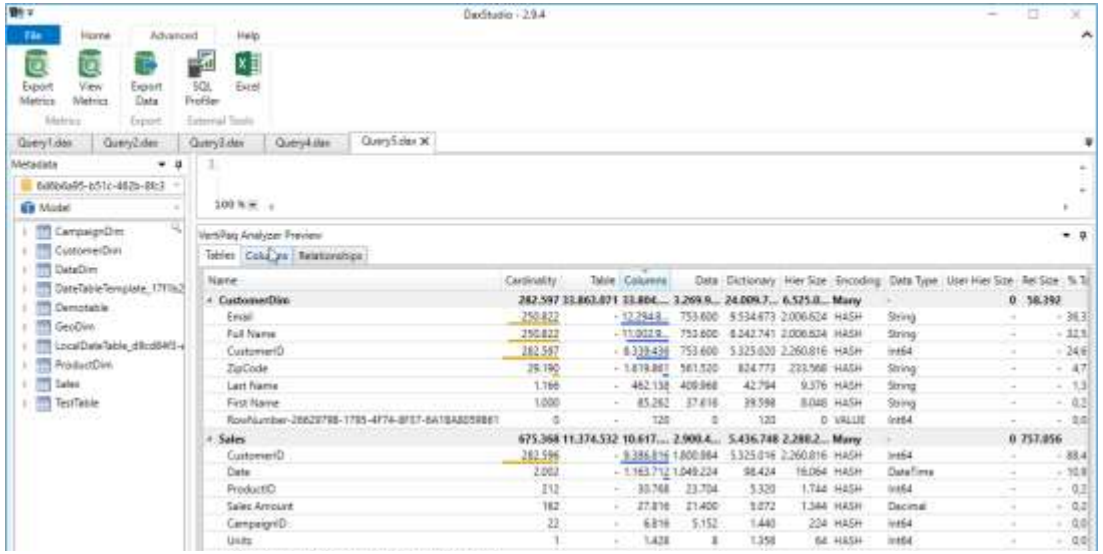
DIAD Complete Data Model

| | |
|---|---:|
| **Data Model** | **59.4 MB** |

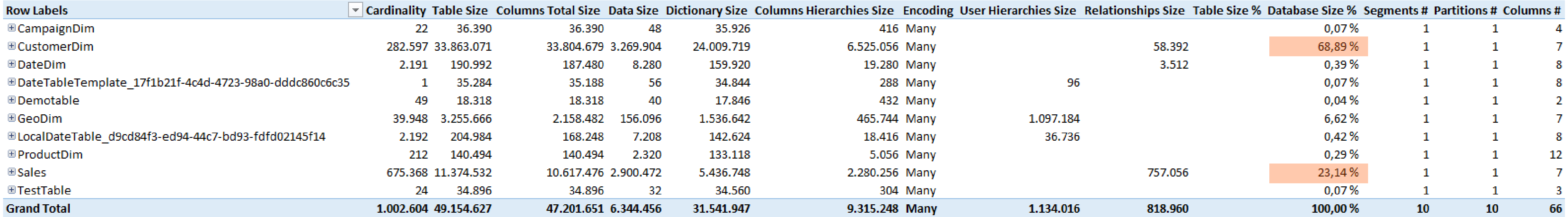
Almost 8X Compression!!

# Data Model can be analyzed with DAX Studio

- Currently in Preview in DAX Studio
  - https://github.com/DaxStudio/DaxStudio
- Available for years in Vertipaq Analyzer
  - https://www.sqlbi.com/tools/vertipaq-analyzer/



| Name | Cardinality | Table | Columns | Data | Dictionary | Hier Size | Encoding | Data Type | User Hier Size | Rel Size | % T |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CustomerDim | 282.597 | 33.863.071 | 33.804.... | 3.269.9... | 24.009.7... | 6.525.0... | Many | - | 0 | 58.392 | |
| Email | 250.822 | - | 12.294.8... | 753.600 | 9.534.673 | 2.006.624 | HASH | String | - | - | 36.3 |
| Full Name | 250.822 | - | 11.002.9... | 753.600 | 8.242.741 | 2.006.624 | HASH | String | - | - | 32.5 |
| CustomerID | 282.597 | - | 8.339.436 | 753.600 | 5.325.020 | 2.260.816 | HASH | Int64 | - | - | 24.6 |
| ZipCode | 29.190 | - | 1.619.861 | 561.520 | 824.773 | 233.568 | HASH | String | - | - | 4.7 |
| Last Name | 1.166 | - | 462.138 | 409.968 | 42.794 | 9.376 | HASH | String | - | - | 1.3 |
| First Name | 1.000 | - | 85.262 | 37.616 | 39.598 | 8.048 | HASH | String | - | - | 0.2 |
| RowNumber-2662D798-1795-4F74-8F57-6A1BA8059B61 | 0 | - | 120 | 0 | 120 | 0 | VALUE | Int64 | - | - | 0.0 |
| Sales | 675.368 | 11.374.532 | 10.617.... | 2.900.4... | 5.436.748 | 2.280.2... | Many | - | 0 | 757.056 | |
| CustomerID | 282.596 | - | 9.386.816 | 1.800.984 | 5.325.016 | 2.260.816 | HASH | Int64 | - | - | 88.4 |
| Date | 2.002 | - | 1.163.712 | 1.049.224 | 98.424 | 16.064 | HASH | DateTime | - | - | 10.9 |
| ProductID | 212 | - | 30.768 | 23.704 | 5.320 | 1.744 | HASH | Int64 | - | - | 0.2 |
| Sales Amount | 182 | - | 27.816 | 21.400 | 5.072 | 1.344 | HASH | Decimal | - | - | 0.2 |
| CampaignID | 22 | - | 6.816 | 5.152 | 1.440 | 224 | HASH | Int64 | - | - | 0.0 |
| Units | 1 | - | 1.428 | 8 | 1.358 | 64 | HASH | Int64 | - | - | 0.0 |



| Row Labels | Cardinality | Table Size | Columns Total Size | Data Size | Dictionary Size | Columns Hierarchies Size | Encoding | User Hierarchies Size | Relationships Size | Table Size % | Database Size % | Segments # | Partitions # | Columns # |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ⊞CampaignDim | 22 | 36.390 | 36.390 | 48 | 35.926 | 416 | Many | | | | 0,07 % | 1 | 1 | 4 |
| ⊞CustomerDim | 282.597 | 33.863.071 | 33.804.679 | 3.269.904 | 24.009.719 | 6.525.056 | Many | | 58.392 | | 68,89 % | 1 | 1 | 7 |
| ⊞DateDim | 2.191 | 190.992 | 187.480 | 8.280 | 159.920 | 19.280 | Many | | 3.512 | | 0,39 % | 1 | 1 | 8 |
| ⊞DateTableTemplate_17f1b21f-4c4d-4723-98a0-dddc860c6c35 | 1 | 35.284 | 35.188 | 56 | 34.844 | 288 | Many | 96 | | | 0,07 % | 1 | 1 | 8 |
| ⊞Demotable | 49 | 18.318 | 18.318 | 40 | 17.846 | 432 | Many | | | | 0,04 % | 1 | 1 | 2 |
| ⊞GeoDim | 39.948 | 3.255.666 | 2.158.482 | 156.096 | 1.536.642 | 465.744 | Many | 1.097.184 | | | 6,62 % | 1 | 1 | 7 |
| ⊞LocalDateTable_d9cd84f3-ed94-44c7-bd93-fdfd02145f14 | 2.192 | 204.984 | 168.248 | 7.208 | 142.624 | 18.416 | Many | 36.736 | | | 0,42 % | 1 | 1 | 8 |
| ⊞ProductDim | 212 | 140.494 | 140.494 | 2.320 | 133.118 | 5.056 | Many | | | | 0,29 % | 1 | 1 | 12 |
| ⊞Sales | 675.368 | 11.374.532 | 10.617.476 | 2.900.472 | 5.436.748 | 2.280.256 | Many | | 757.056 | | 23,14 % | 1 | 1 | 7 |
| ⊞TestTable | 24 | 34.896 | 34.896 | 32 | 34.560 | 304 | Many | | | | 0,07 % | 1 | 1 | 3 |
| **Grand Total** | **1.002.604** | **49.154.627** | **47.201.651** | **6.344.456** | **31.541.947** | **9.315.248** | **Many** | **1.134.016** | **818.960** | | **100,00 %** | **10** | **10** | **66** |

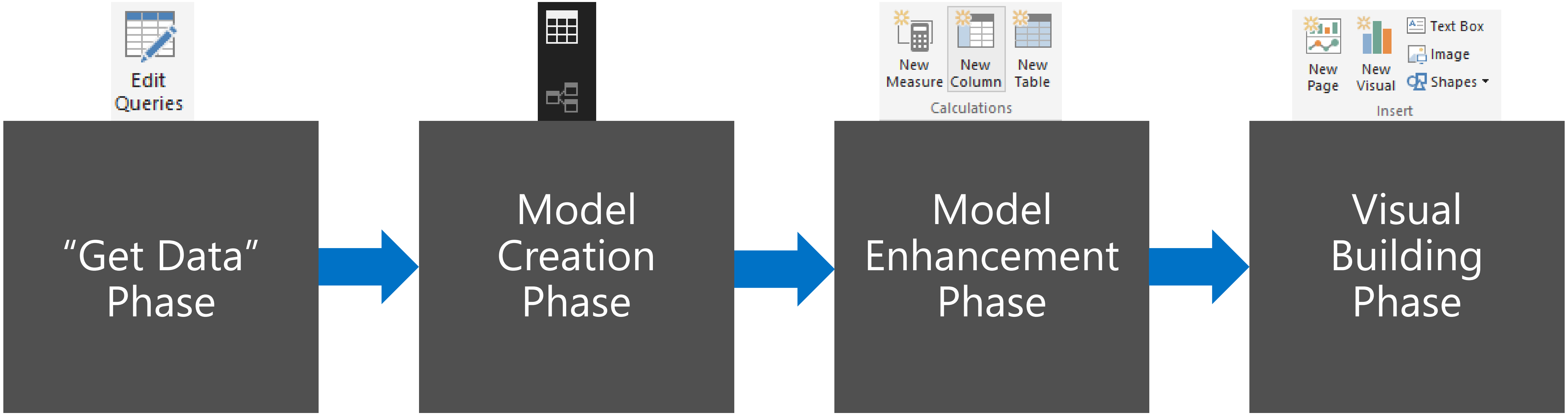
avanade

# History of Power BI

- 2009 Gemini → Power Pivot (Excel – InMemory data model)
- 2012 Power View (Excel SharePoint – Visualizations)
- 2012 Analysis Services tabular (SQL Server – inMemory engine to complement multidimensional)
- 2013 Data Explorer → Power Query (Excel – Data Connectivity)
- 2015 Power BI (all in one solution including hosted service)

# Power BI Desktop Files

## Phases in Building a Power BI Desktop File



"Get Data" Phase

Model Creation Phase

Model Enhancement Phase

Visual Building Phase

Create Query in "M"

Compress data, auto detect relationships

(Automatically done by Tool)

Add calculated columns, measures, add missing relationships

Evaluate Measures and build each visual

# Designing good data models

Power BI

## Key takeaways to design a good Power BI Desktop data model

- **RAM is precious !!!!!**

## Some Tips and tricks to save RAM and increase speed of model

- If a fact table contains an ID field which is unique for each record, **remove it**
  - Ex. Transaction ID

- **Sort columns** before bringing them into a Power BI data model

- The DateTime data type is usually not needed, unless you are specifically using the Time component
  - ➢ If you really need Time, **try splitting Date & Time** into two columns - Reduces # of unique values

**Star Schema** – Good for most Data Models

# Data Types

Power BI

## Numeric Data Types

- Whole Number
- Decimal Number
- Fixed Decimal Number (Floating point stored as integer)
- Boolean

## Date/Time Data Types

- Date – Internally stored as an integer
- Time – Internally stored as a fraction between 0 and 1
- Date Time

## Other Data Types

- Text

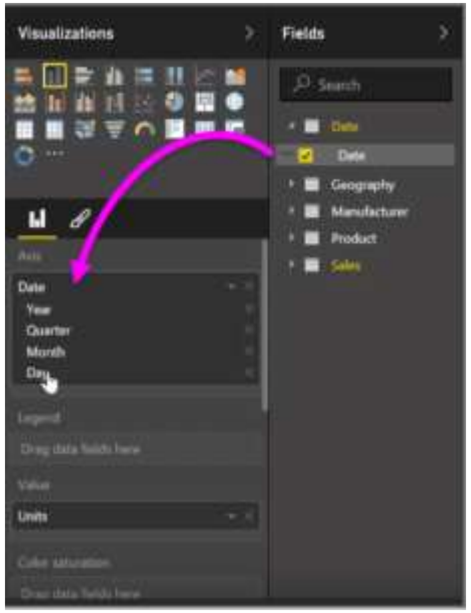
*Set your*
***Data Types***
*in the*
*Query Editor*

*Set your*
***Data Formats***
*($ %, etc)*
*in the Data Model*

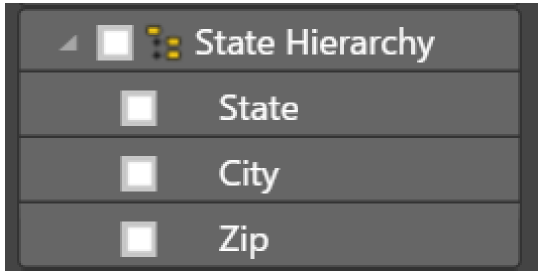
**Pro Tip: Data type is different from data format**

# Hierarchies

- Power BI generates Date hierarchies when dates are added to visuals, this allows the end user to drill from Year, Quarter, Month & Day.

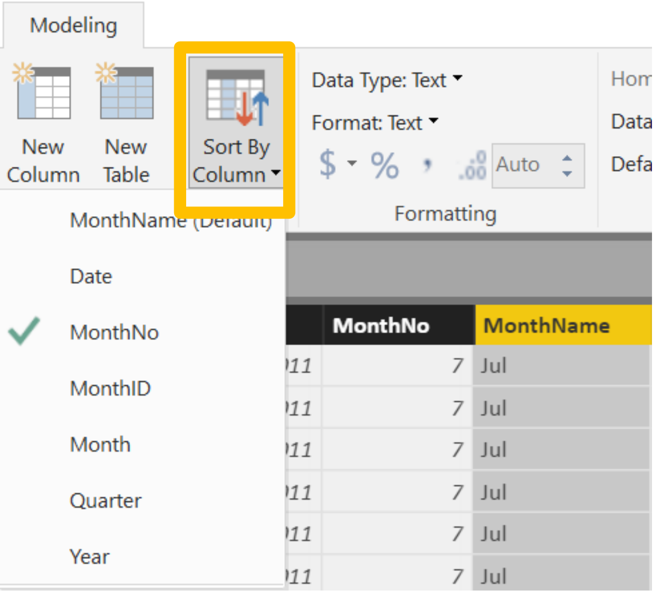


- Users can also create custom hierarchies in the model by dragging a lower level field onto the parent.



# Sort By Column

- Enables sorting one text field by another (numeric) field
- Contradictions will cause an error while loading
- Needed for example for months



| MonthNo | MonthName |
|---|---|
| 7 | Jul |
| 7 | Jul |
| 7 | Jul |
| 7 | Jul |
| 7 | Jul |
| 7 | Jul |

# KNOWLEDGE CHECK Module

Power BI

1. What is a *data model* in the context of Power BI?
2. What are some advantages of a star schema over a flat or denormalized model?
3. How might you improve the performance of a Power BI model?
4. How does Power BI store DateTime information? What are some consequences of this? How should DateTime be modelled

# KNOWLEDGE CHECK ANSWERS Module

- What is a *data model* in the context of Power BI?
  - *A data model is a collection of tables and relationships*

- What are some advantages of a star schema over a flat or denormalized model?
  - *Dimension tables save space by reducing the amount of data that needs to be repeated over and over in every row*
  - *Relationships between tables can be leveraged for more complex measures*

- How might you improve the performance of a Power BI model?
  - *Try using a star schema instead of a flat or denormalized model*
  - *Remove unnecessary columns*
  - *Set appropriate data types*

- How does Power BI store DateTime information? What are some consequences of this?
  - *DateTime information is stored as a floating-point decimal number. This means that datetimes are very precise but not very efficient to store.*

# Download example pbix

http://aka.avanade.com/ppwt2019pbix

avanade

# Module Lab

1. Open up the file **Student Modeling Pre-class.pbix**

2. Create the relationships between the tables!
   HINT: You may need to preview some of the tables to see what is in them

   Think about: What sort of data model are you creating?

# Agenda

|  |  |  |  |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
|  | ***Mid Morning break*** |  | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
|  | ***Lunch*** |  | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
|  | ***Afternoon break*** |  | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Module

# DAX Calculated Columns & Measures

# MODULE OBJECTIVE

- Understand differences between calculated columns and measures (uses, evaluation, performance, etc.)

# DAX Level Set

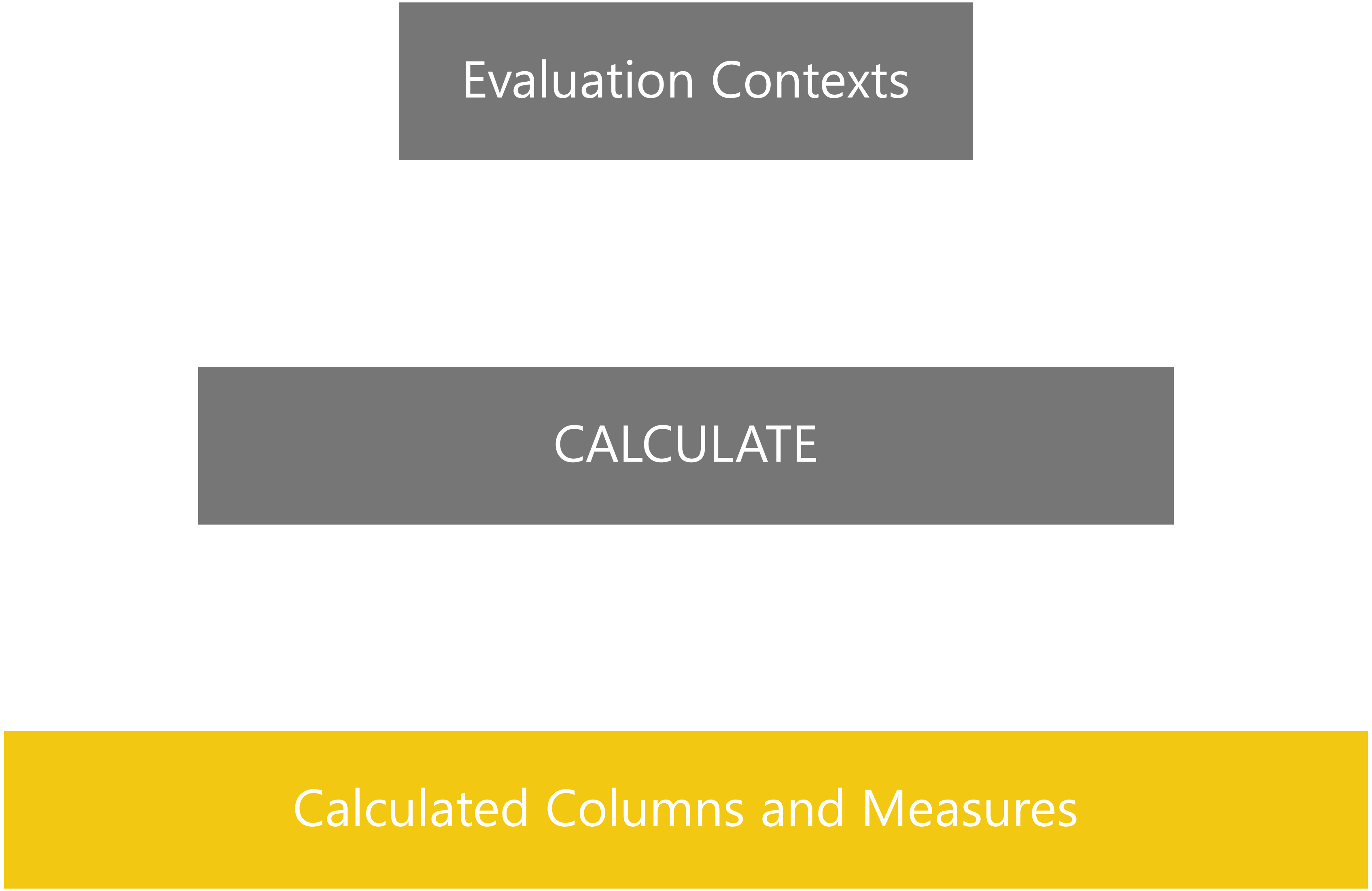
- DAX looks similar to Excel functions, but they have key differences

- DAX is a very deep and elegant…

- This class provides a solid base in DAX, but don’t expect to leave being able to write the most complex DAX patterns – they take practice.

# DAX Foundations

## Path to DAX Expertise



Evaluation Contexts

CALCULATE

Calculated Columns and Measures

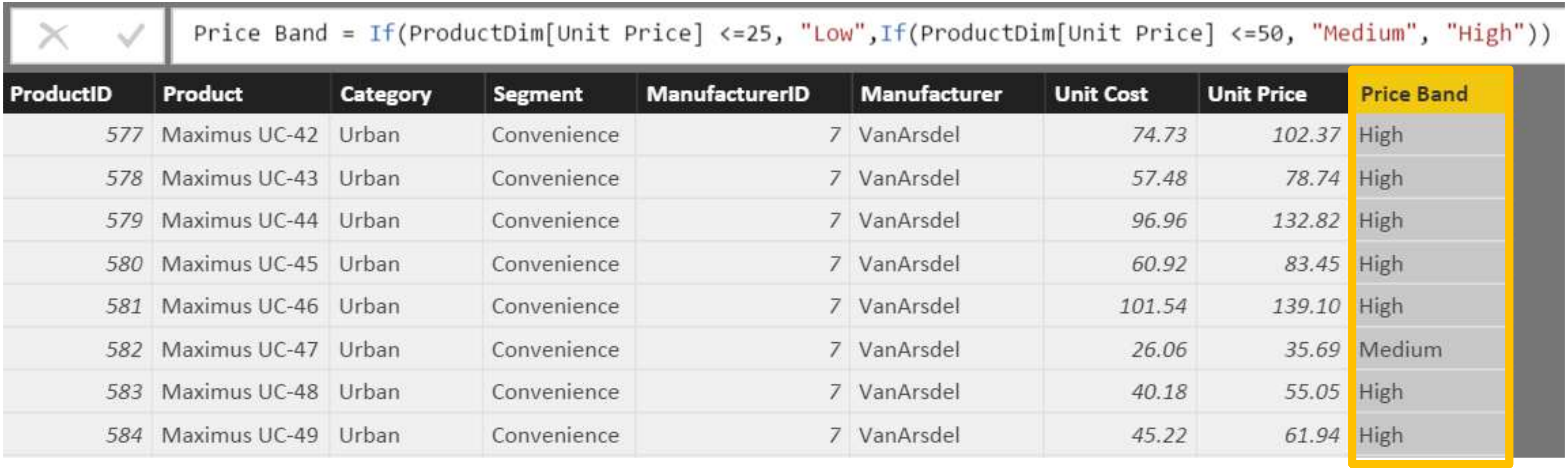
# DAX Foundations

Power BI

Calculated Column

Measure

# What is a Calculated Column?



```
Price Band = If(ProductDim[Unit Price] <=25, "Low",If(ProductDim[Unit Price] <=50, "Medium", "High"))
```

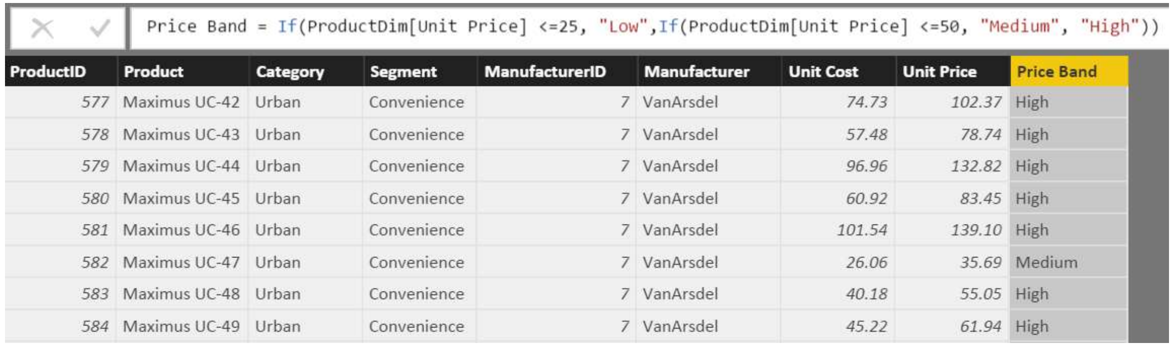
| ProductID | Product | Category | Segment | ManufacturerID | Manufacturer | Unit Cost | Unit Price | Price Band |
|---|---|---|---|---|---|---|---|---|
| 577 | Maximus UC-42 | Urban | Convenience | 7 | VanArsdel | 74.73 | 102.37 | High |
| 578 | Maximus UC-43 | Urban | Convenience | 7 | VanArsdel | 57.48 | 78.74 | High |
| 579 | Maximus UC-44 | Urban | Convenience | 7 | VanArsdel | 96.96 | 132.82 | High |
| 580 | Maximus UC-45 | Urban | Convenience | 7 | VanArsdel | 60.92 | 83.45 | High |
| 581 | Maximus UC-46 | Urban | Convenience | 7 | VanArsdel | 101.54 | 139.10 | High |
| 582 | Maximus UC-47 | Urban | Convenience | 7 | VanArsdel | 26.06 | 35.69 | Medium |
| 583 | Maximus UC-48 | Urban | Convenience | 7 | VanArsdel | 40.18 | 55.05 | High |
| 584 | Maximus UC-49 | Urban | Convenience | 7 | VanArsdel | 45.22 | 61.94 | High |

**Calculated Column**

**Pro Tip:** Always refer to a calculated column by its full name -> **TableName[ColumnName]**

# Calculated Column

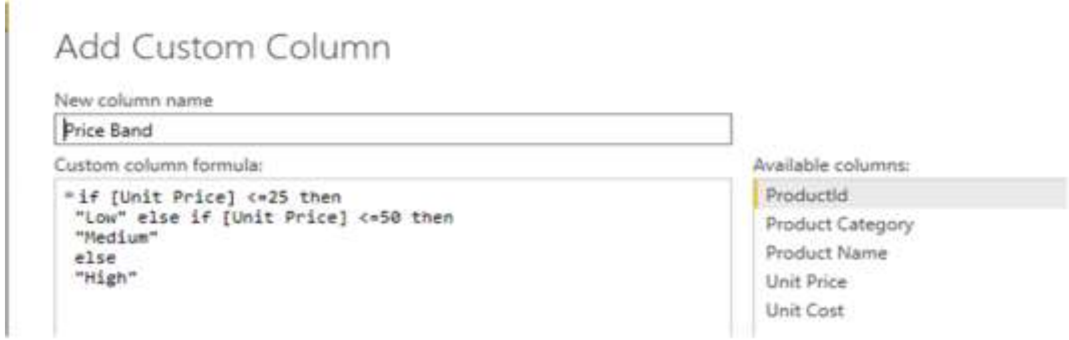
## Calculated Column in DAX



```
Price Band = If(ProductDim[Unit Price] <=25, "Low",If(ProductDim[Unit Price] <=50, "Medium", "High"))
```

| ProductID | Product | Category | Segment | ManufacturerID | Manufacturer | Unit Cost | Unit Price | Price Band |
|---|---|---|---|---|---|---|---|---|
| 577 | Maximus UC-42 | Urban | Convenience | 7 | VanArsdel | 74.73 | 102.37 | High |
| 578 | Maximus UC-43 | Urban | Convenience | 7 | VanArsdel | 57.48 | 78.74 | High |
| 579 | Maximus UC-44 | Urban | Convenience | 7 | VanArsdel | 96.96 | 132.82 | High |
| 580 | Maximus UC-45 | Urban | Convenience | 7 | VanArsdel | 60.92 | 83.45 | High |
| 581 | Maximus UC-46 | Urban | Convenience | 7 | VanArsdel | 101.54 | 139.10 | High |
| 582 | Maximus UC-47 | Urban | Convenience | 7 | VanArsdel | 26.06 | 35.69 | Medium |
| 583 | Maximus UC-48 | Urban | Convenience | 7 | VanArsdel | 40.18 | 55.05 | High |
| 584 | Maximus UC-49 | Urban | Convenience | 7 | VanArsdel | 45.22 | 61.94 | High |

## Custom Column in “Query Editor”



Add Custom Column

New column name

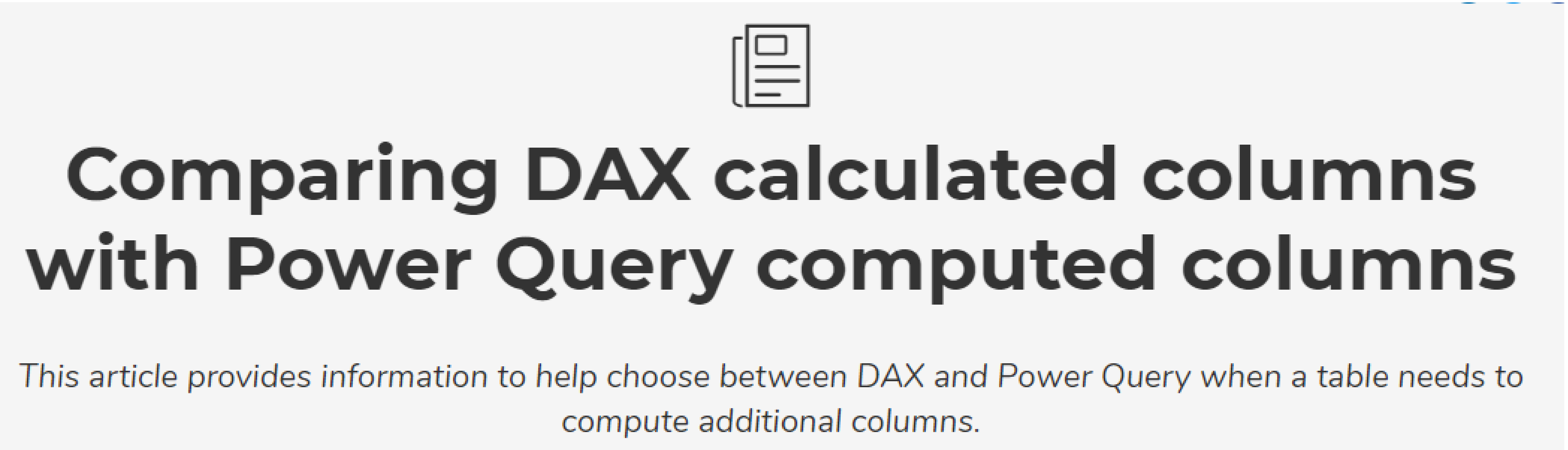
Price Band

Custom column formula:

```
=if [Unit Price] <=25 then
 "Low" else if [Unit Price] <=50 then
 "Medium"
 else
 "High"
```

**Note:** If given a choice, creating the column in “M” or “Query Editor” will give you better compression.

# More Details on this topic



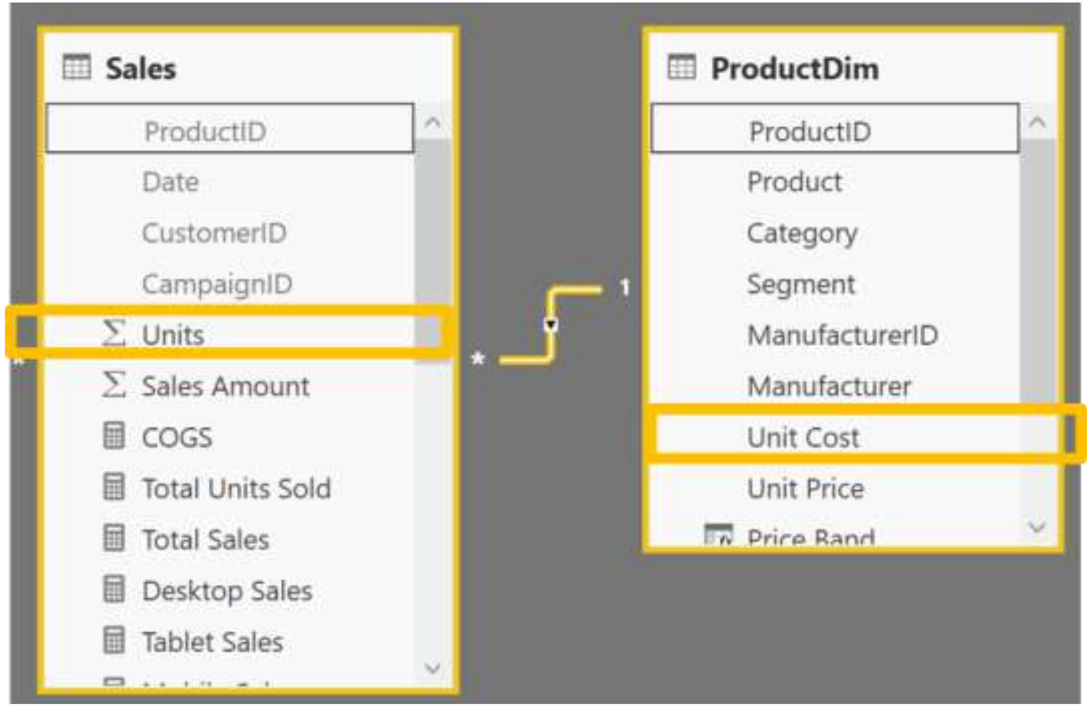
## Comparing DAX calculated columns with Power Query computed columns

*This article provides information to help choose between DAX and Power Query when a table needs to compute additional columns.*

https://sql.bi/439946

# Calculated Column

## Calculated Column – Accessing columns from other Tables in model

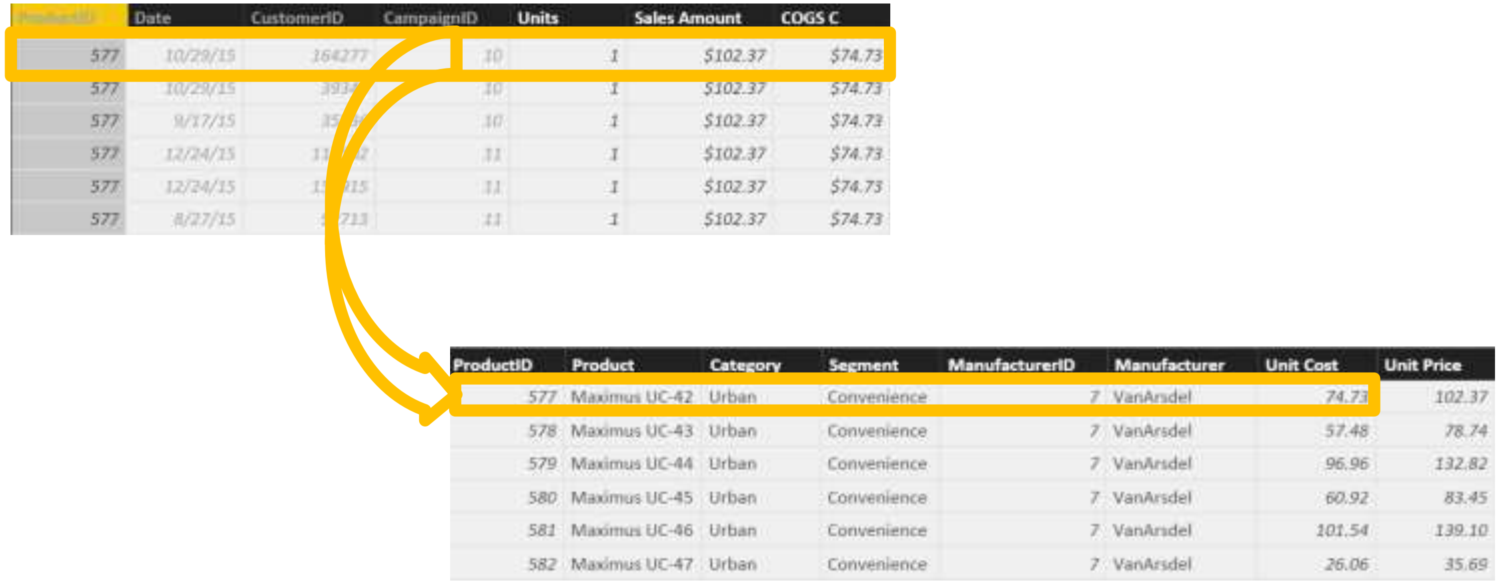


- Often you want to access columns from multiple tables to create a Calculated Columns
- Let us say you want to calculate COGs, which is Units * Units Cost
- Units Cost is in another Table

# RELATED Function

## Row Context and Multiple Tables – RELATED Function

Sales[COGS] = RELATED(ProductDim[Unit Cost]) * Sales[Units]



| ProductID | Date | CustomerID | CampaignID | Units | Sales Amount | COGS C |
|---|---|---|---|---|---|---|
| 577 | 10/29/15 | 164277 | 10 | 1 | $102.37 | $74.73 |
| 577 | 10/29/15 | 3934[illegible] | 10 | 1 | $102.37 | $74.73 |
| 577 | 9/17/15 | 35[illegible] | 10 | 1 | $102.37 | $74.73 |
| 577 | 12/24/15 | 11[illegible]2 | 11 | 1 | $102.37 | $74.73 |
| 577 | 12/24/15 | 1[illegible]15 | 11 | 1 | $102.37 | $74.73 |
| 577 | 8/27/15 | [illegible]713 | 11 | 1 | $102.37 | $74.73 |

| ProductID | Product | Category | Segment | ManufacturerID | Manufacturer | Unit Cost | Unit Price |
|---|---|---|---|---|---|---|---|
| 577 | Maximus UC-42 | Urban | Convenience | 7 | VanArsdel | 74.73 | 102.37 |
| 578 | Maximus UC-43 | Urban | Convenience | 7 | VanArsdel | 57.48 | 78.74 |
| 579 | Maximus UC-44 | Urban | Convenience | 7 | VanArsdel | 96.96 | 132.82 |
| 580 | Maximus UC-45 | Urban | Convenience | 7 | VanArsdel | 60.92 | 83.45 |
| 581 | Maximus UC-46 | Urban | Convenience | 7 | VanArsdel | 101.54 | 139.10 |
| 582 | Maximus UC-47 | Urban | Convenience | 7 | VanArsdel | 26.06 | 35.69 |

- RELATED is just like VLOOKUP in Excel

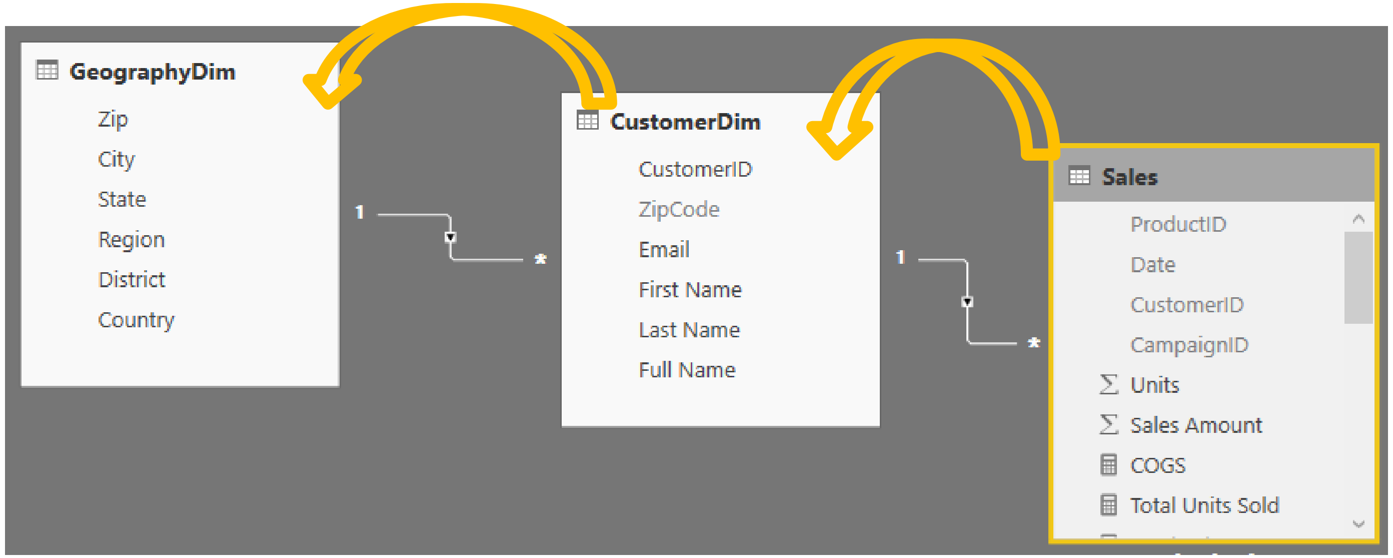
# RELATED Function

Power BI

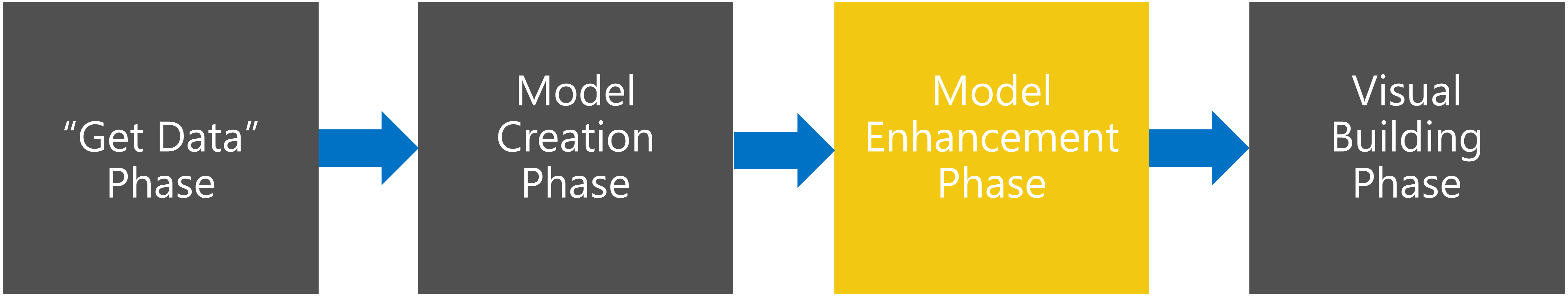
## RELATED Function Example

- You *could* follow a chain of relationship from Many side to 1 side using RELATED

**Sales [City State]= RELATED(GeographyDim[City]) & “, ” & RELATED(GeographyDim[State])**



## When is a Calculated Column Evaluated?



- Create Query in "M"

- Compress data
- Auto detect relationships

- Add calc. columns, Measures
- Add missing relationships

- Evaluate Measures and build each visual

# Best Practices – Calculated Columns

## Best Practices with DAX Calculated Columns

- Whenever possible, DAX helper columns should be avoided. Each “Helper Column” will consume RAM
- Create a calculated column in the Dim Table as opposed to in the Fact Table
- Move calculated columns to “M” if you can
- Calculated Columns are evaluated during processing

# DAX Foundations

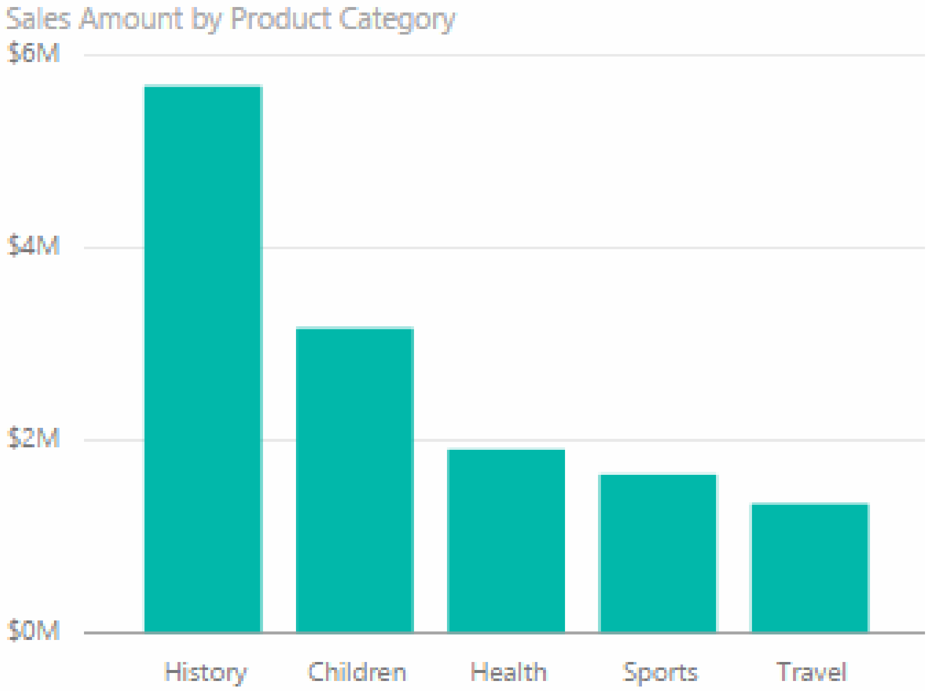
Power BI

Calculated Column

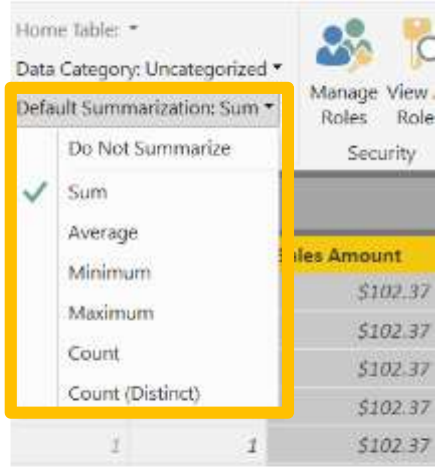
Measure

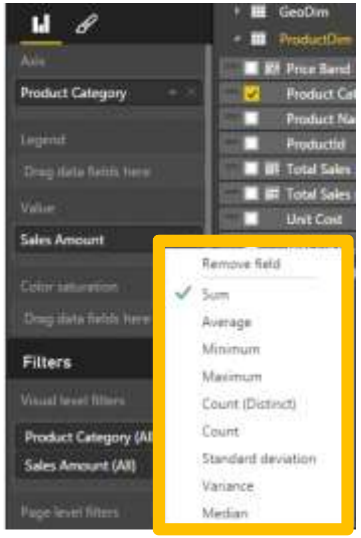
# Default Summarization

## What is a Default Summarization?



On the Modeling ribbon



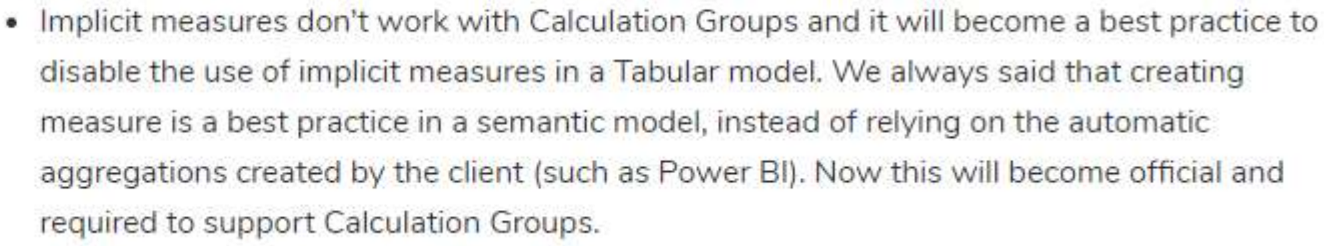


For an individual visualization

- Sales Amount is automatically summed for each category
- You should set this summarization for all numeric fields in the **Modeling** ribbon

# Never use Implicit Measures

- It is considered best practice to not use implicit measures



- Implicit measures don't work with Calculation Groups and it will become a best practice to disable the use of implicit measures in a Tabular model. We always said that creating measure is a best practice in a semantic model, instead of relying on the automatic aggregations created by the client (such as Power BI). Now this will become official and required to support Calculation Groups.

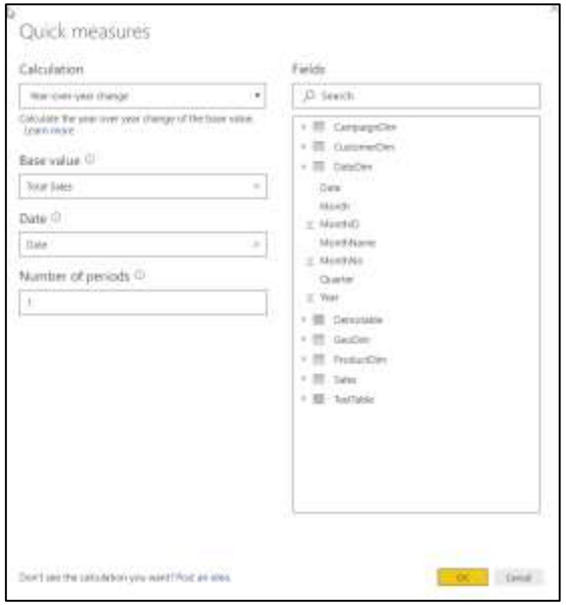
- Excel does not work with implicit measures

# Quick Measures

Power BI

## Quick Measures are wizard driven DAX calculations

- Right+Click a table and select **Quick Measures**
  - Select calculation type and fill-in parameters
  - DAX is generated automatically
  - Great way to learn DAX



Quick measures

Calculation: Year-over-year change

Calculate the year over year change of the base value. Learn more

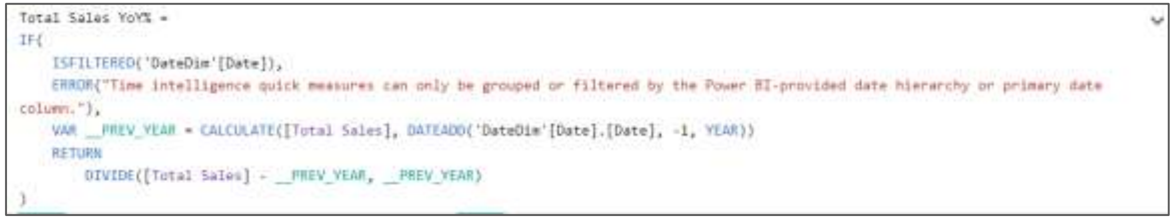
Base value: Total Sales

Date: Date

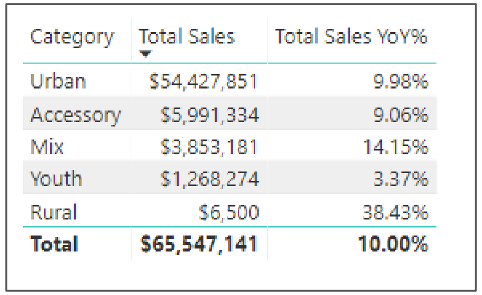
Number of periods: 1

Fields

Don't see the calculation you want? Post an idea.



```
Total Sales YoY% =
IF(
    ISFILTERED('DateDim'[Date]),
    ERROR("Time intelligence quick measures can only be grouped or filtered by the Power BI-provided date hierarchy or primary date
column."),
    VAR __PREV_YEAR = CALCULATE([Total Sales], DATEADD('DateDim'[Date].[Date], -1, YEAR))
    RETURN
        DIVIDE([Total Sales] - __PREV_YEAR, __PREV_YEAR)
)
```

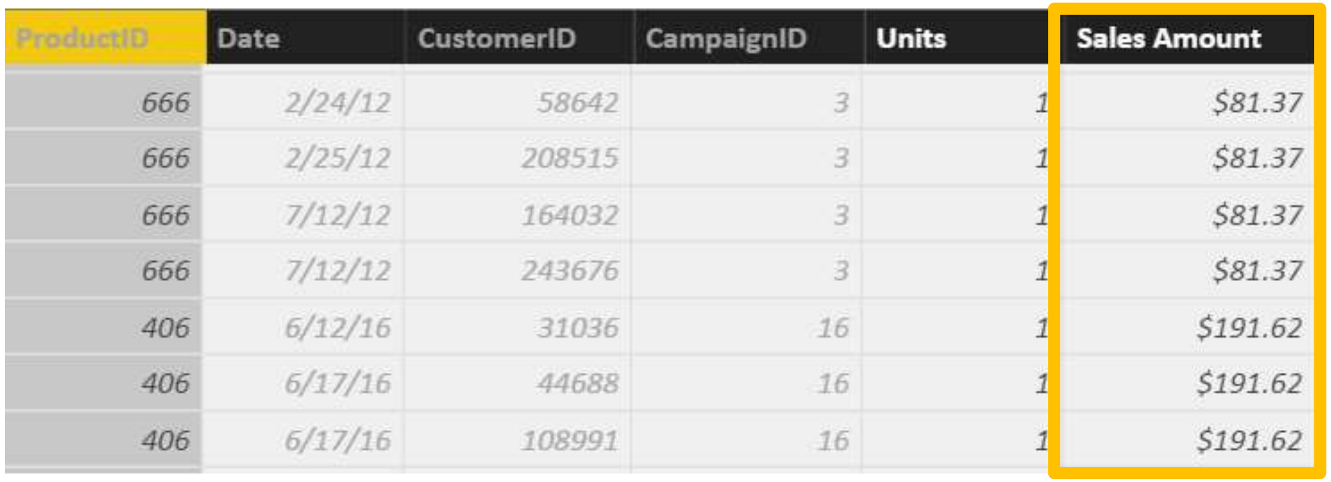


| Category | Total Sales | Total Sales YoY% |
|---|---|---|
| Urban | $54,427,851 | 9.98% |
| Accessory | $5,991,334 | 9.06% |
| Mix | $3,853,181 | 14.15% |
| Youth | $1,268,274 | 3.37% |
| Rural | $6,500 | 38.43% |
| **Total** | **$65,547,141** | **10.00%** |

See Quick Measure gallery: https://community.powerbi.com/t5/Quick-Measures-Gallery/bd-p/QuickMeasuresGallery

# Measures

## What is a Measure?



| ProductID | Date | CustomerID | CampaignID | Units | Sales Amount |
|---|---|---|---|---|---|
| 666 | 2/24/12 | 58642 | 3 | 1 | $81.37 |
| 666 | 2/25/12 | 208515 | 3 | 1 | $81.37 |
| 666 | 7/12/12 | 164032 | 3 | 1 | $81.37 |
| 666 | 7/12/12 | 243676 | 3 | 1 | $81.37 |
| 406 | 6/12/16 | 31036 | 16 | 1 | $191.62 |
| 406 | 6/17/16 | 44688 | 16 | 1 | $191.62 |
| 406 | 6/17/16 | 108991 | 16 | 1 | $191.62 |

**[Total Sales]=SUM(Sales[Sales Amount])**

- Measures are created using DAX
- Place your Measures on a Fact table for best results

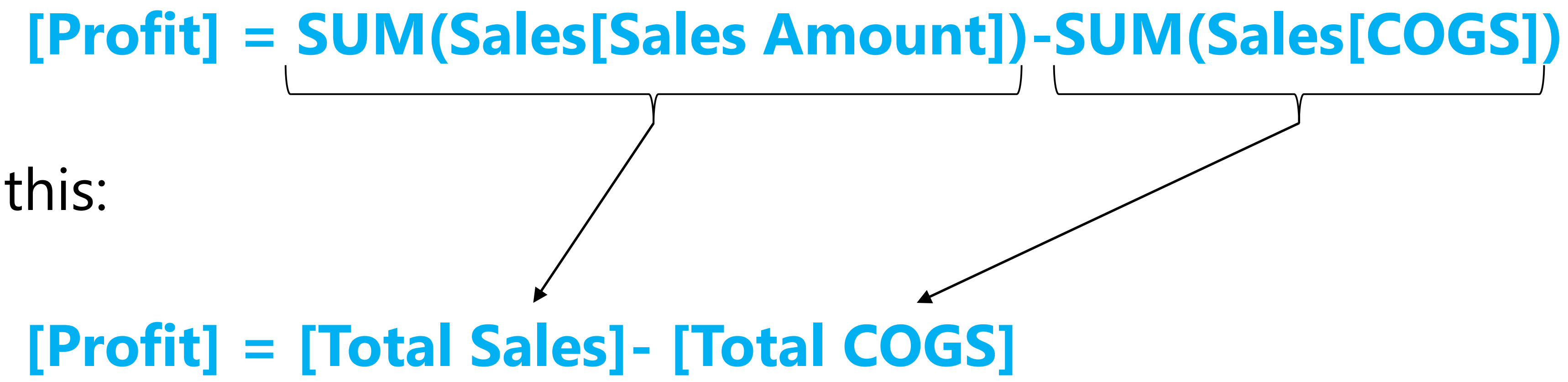
**Pro Tip:** When referring to a measure in other calculations, refer to it without a Table name: **[MeasureName]**

# Measures

Power BI

## Measure, Use Case 1: Using One Measure in Another

Instead of writing this:



[Profit] = SUM(Sales[Sales Amount])-SUM(Sales[COGS])

Write this:

[Profit] = [Total Sales]- [Total COGS]

- Allows re-use of measures
- Formulas are much simpler to read

# Measures

## Measure, Use Case 2: More Complex Calculations

**[Profit Margin %] = [Profit] / [Total Sales]**

- Ratios are calculations that cannot be created using a Calculated Column or Default Summarization
- Use DAX **DIVIDE** for built in error handling

**[Profit Margin %] = DIVIDE([Profit] , [Total Sales])**

# Think about the order of aggregation!

| Customer | Tons | Price | Price Per Ton |
|---|---|---|---|
| C1 | 5 | 10,00 € | 2,00 € |
| C2 | 20 | 20,00 € | 1,00 € |
| C1 | 5 | 12,00 € | 2,40 € |
| C3 | 10 | 5,00 € | 0,50 € |
| **Total** | **40** | **47,00 €** | **???** |

**Row-wise**
Option 1: 2 + 1+ 2,4 + 0,5 = 5,90 EUR
Option 2: AVG(2 ; 1; 2,4 ; 0,5) = 1,48 EUR
**First aggregate**
Option 3: 47 EUR / 40 = 1,18 EUR

# Measures

Power BI

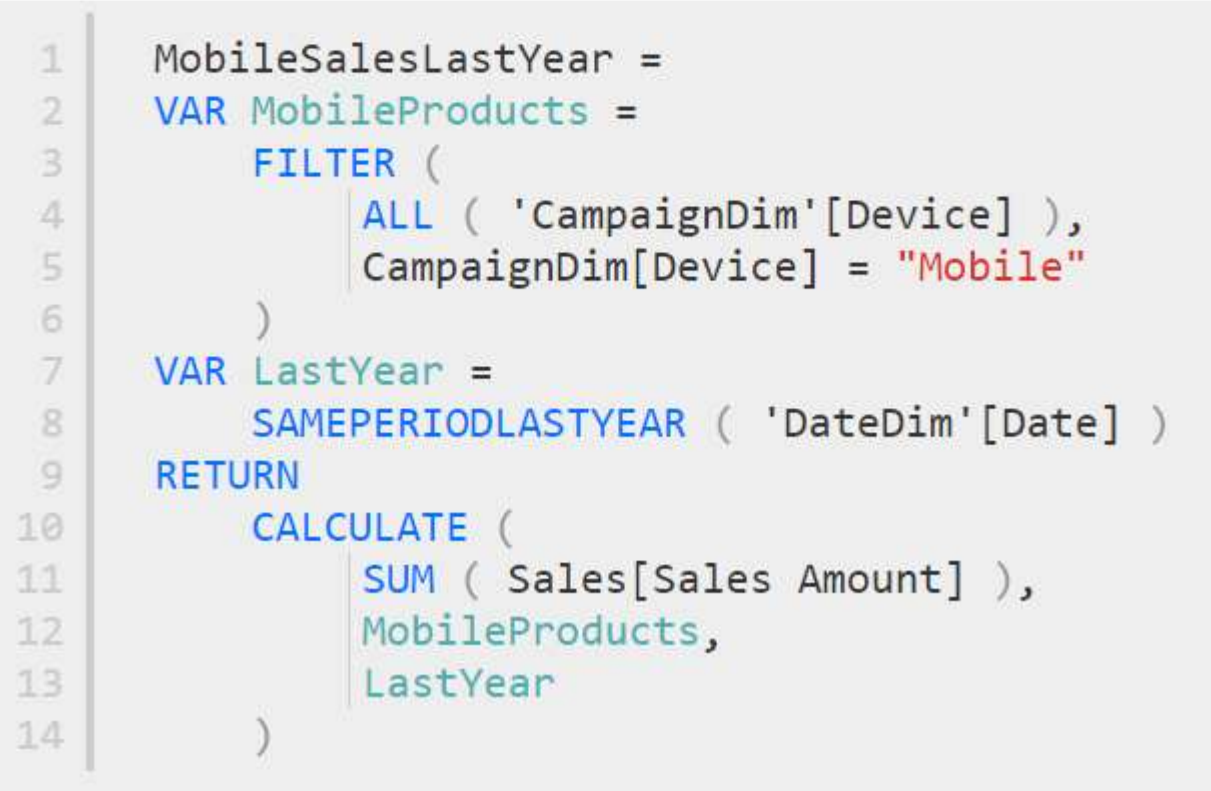
## Measure, Use Case 3: More Complex Calculations Using Variables

```
MobileSalesLastYear =
   VAR MobileProducts = FILTER(
                    ALL('CampaignDim'[Device]),
                    CampaignDim[Device]="Mobile"
               )
   VAR LastYear = SAMEPERIODLASTYEAR('DateDim'[Date])
   RETURN
   CALCULATE(SUM(Sales[Sales
Amount]),MobileProducts,LastYear)
```

- Allows re-use of variables
- Formulas are much simpler to read

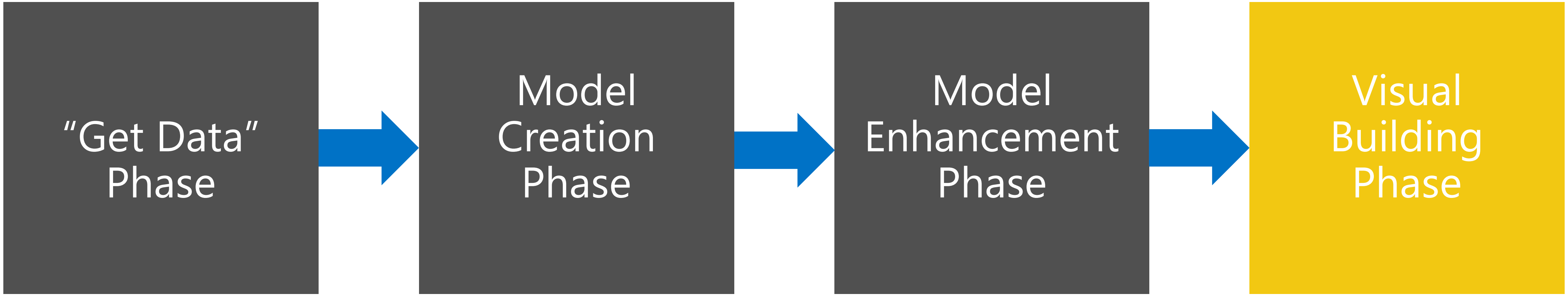
# Dax Formatter makes measures easier to read

- Formats any DAX measure
- https://www.daxformatter.com/



```
MobileSalesLastYear =
VAR MobileProducts =
    FILTER (
        ALL ( 'CampaignDim'[Device] ),
        CampaignDim[Device] = "Mobile"
    )
VAR LastYear =
    SAMEPERIODLASTYEAR ( 'DateDim'[Date] )
RETURN
    CALCULATE (
        SUM ( Sales[Sales Amount] ),
        MobileProducts,
        LastYear
    )
```

## When is a Measure Evaluated?



"Get Data" Phase

Model Creation Phase

Model Enhancement Phase

Visual Building Phase

**"Get Data" Phase**

- Create Query in "M"

**Model Creation Phase**

- Compress data
- auto detect relationships

**Model Enhancement Phase**

- Add calc. columns, Measures
- Add missing relationships

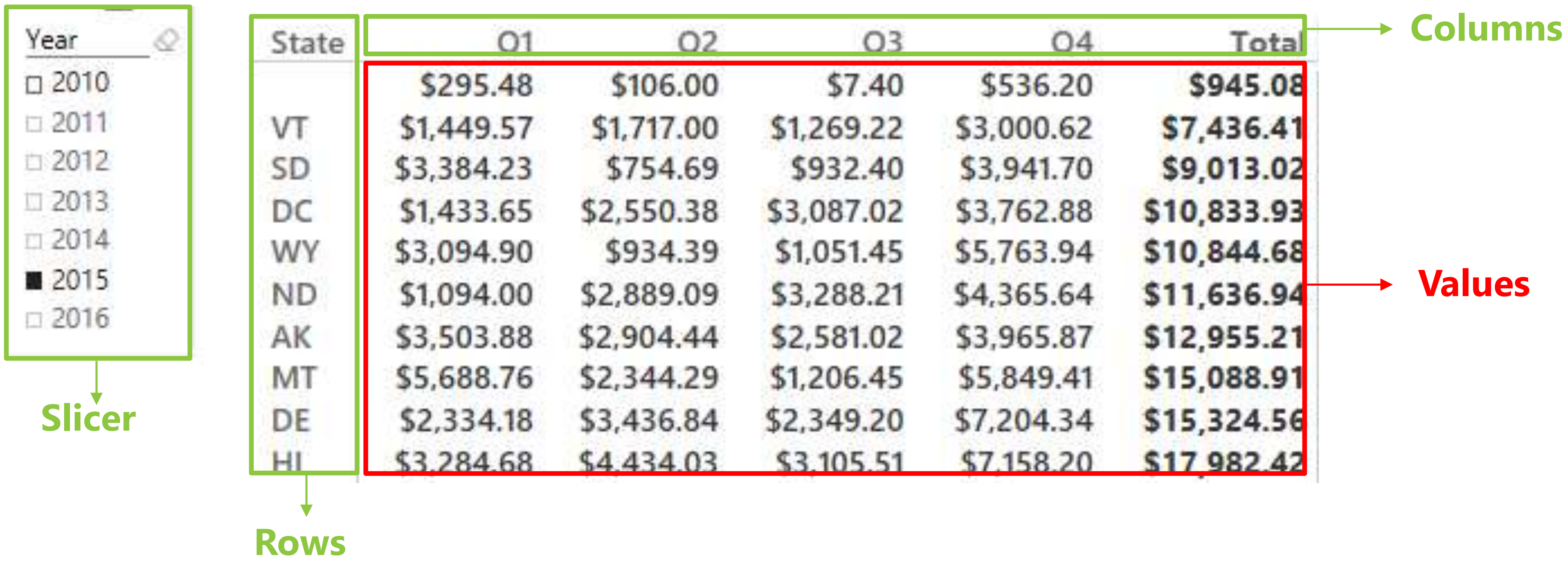
**Visual Building Phase**

- Evaluate Measures and build each visual

# Calculated Column vs. Measure

Power BI

## Calculated Column vs. Measure - When to Use What



Year

□ 2010
□ 2011
□ 2012
□ 2013
□ 2014
■ 2015
□ 2016

| State | Q1 | Q2 | Q3 | Q4 | Total |
|---|---|---|---|---|---|
| | $295.48 | $106.00 | $7.40 | $536.20 | $945.08 |
| VT | $1,449.57 | $1,717.00 | $1,269.22 | $3,000.62 | $7,436.41 |
| SD | $3,384.23 | $754.69 | $932.40 | $3,941.70 | $9,013.02 |
| DC | $1,433.65 | $2,550.38 | $3,087.02 | $3,762.88 | $10,833.93 |
| WY | $3,094.90 | $934.39 | $1,051.45 | $5,763.94 | $10,844.68 |
| ND | $1,094.00 | $2,889.09 | $3,288.21 | $4,365.64 | $11,636.94 |
| AK | $3,503.88 | $2,904.44 | $2,581.02 | $3,965.87 | $12,955.21 |
| MT | $5,688.76 | $2,344.29 | $1,206.45 | $5,849.41 | $15,088.91 |
| DE | $2,334.18 | $3,436.84 | $2,349.20 | $7,204.34 | $15,324.56 |
| HI | $3,284.68 | $4,434.03 | $3,105.51 | $7,158.20 | $17,982.42 |

Columns

Values

Slicer

Rows

### Rule of Thumb for Calculated Column vs Measure

- **Calculated Column** – Use in Page, Report & Visual Filters as well as Slicers, Rows and Columns
- **Measures** - Use in Values section

# KNOWLEDGE CHECK Module

- When is Calculated Column Evaluated?
- What is Default Summarization?
- When is a Measure Evaluated?
- When to use Measures and Calculated Columns?

# KNOWLEDGE CHECK ANSWERS Module

Power BI

- When is Calculated Column Evaluated?
  - *At the time of data load/data refresh.*

- What is Default Summarization?
  - *A default summarization is an implicit measure created in the background when you put a numeric field on a visualization. The function used (sum/max/min/avg/…) is based on the numeric field's default summarization setting.*

- When is a Measure Evaluated?
  - *At render time.*

- When to use Measures and Calculated Columns?
  - *It depends ☺. Calculated columns are useful when each row of data should be independently considered (although measures can do this too!) and the result won't change until the next data refresh. Measures should be used everywhere else.*

# Module Lab

1. Create a MEASURE for Total Units Sold
   HINT: The formula will probably use SUM()

2. Create a CALCULATED COLUMN on the fact table that shows product category and campaign traffic channel combined
   *Example*: Urban, Organic Search

3. It is easy to see that the CALCULATED COLUMN is working. Create some visuals that allow you to confirm that the Total Units Sold MEASURE is working right

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Module
# CALCULATE

# MODULE 3 OBJECTIVE

- Understand the basics of the CALCULATE formula

# DAX Foundations

## PATH to DAX Expertise

Evaluation Contexts

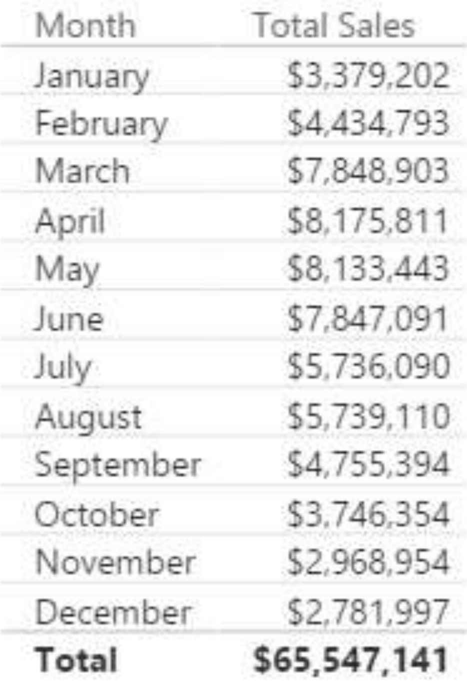
CALCULATE

Calculated Columns and Measures

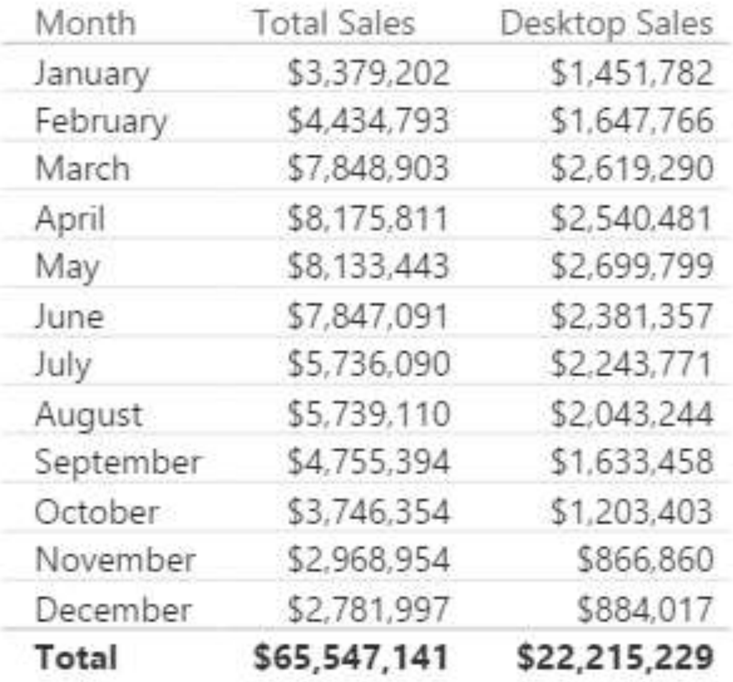
## Why is CALCULATE Useful?

You create a report of
breakdown of Sales by Month

**<u>Typical Business Question:</u>**
Provide a break out of this Sales from Desktop



| Month | Total Sales |
|---|---|
| January | $3,379,202 |
| February | $4,434,793 |
| March | $7,848,903 |
| April | $8,175,811 |
| May | $8,133,443 |
| June | $7,847,091 |
| July | $5,736,090 |
| August | $5,739,110 |
| September | $4,755,394 |
| October | $3,746,354 |
| November | $2,968,954 |
| December | $2,781,997 |
| **Total** | **$65,547,141** |



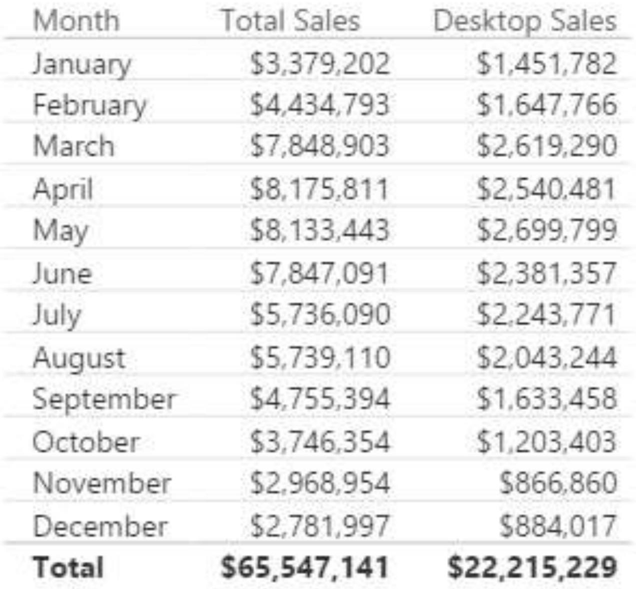
| Month | Total Sales | Desktop Sales |
|---|---|---|
| January | $3,379,202 | $1,451,782 |
| February | $4,434,793 | $1,647,766 |
| March | $7,848,903 | $2,619,290 |
| April | $8,175,811 | $2,540,481 |
| May | $8,133,443 | $2,699,799 |
| June | $7,847,091 | $2,381,357 |
| July | $5,736,090 | $2,243,771 |
| August | $5,739,110 | $2,043,244 |
| September | $4,755,394 | $1,633,458 |
| October | $3,746,354 | $1,203,403 |
| November | $2,968,954 | $866,860 |
| December | $2,781,997 | $884,017 |
| **Total** | **$65,547,141** | **$22,215,229** |

# CALCULATE

## Here is how you do it with CALCULATE

**[Desktop Sales] = CALCULATE([Total Sales], CampaignDim[Device] = "Desktop")**

- Use CALCULATE function to create a Measure which filters down to Desktop Sales



| Month | Total Sales | Desktop Sales |
|---|---|---|
| January | $3,379,202 | $1,451,782 |
| February | $4,434,793 | $1,647,766 |
| March | $7,848,903 | $2,619,290 |
| April | $8,175,811 | $2,540,481 |
| May | $8,133,443 | $2,699,799 |
| June | $7,847,091 | $2,381,357 |
| July | $5,736,090 | $2,243,771 |
| August | $5,739,110 | $2,043,244 |
| September | $4,755,394 | $1,633,458 |
| October | $3,746,354 | $1,203,403 |
| November | $2,968,954 | $866,860 |
| December | $2,781,997 | $884,017 |
| **Total** | **$65,547,141** | **$22,215,229** |

# CALCULATE

## Anatomy of CALCULATE

**CALCULATE(Expression, [Filter 1], [Filter 2]…..)**

**Filter Arguments**

- EXPRESSION used as the first parameter is essentially the same as a measure
- CALCULATE works differently from other DAX functions
- The second set of arguments, i.e. the “Filter arguments,” are evaluated and applied first
- Then the Expression is evaluated under new “Filter Context"

# CALCULATE – Add Filter

Power BI

## CALCULATE – The Most Important Function in DAX

4 Key functions that CALCULATE can do:

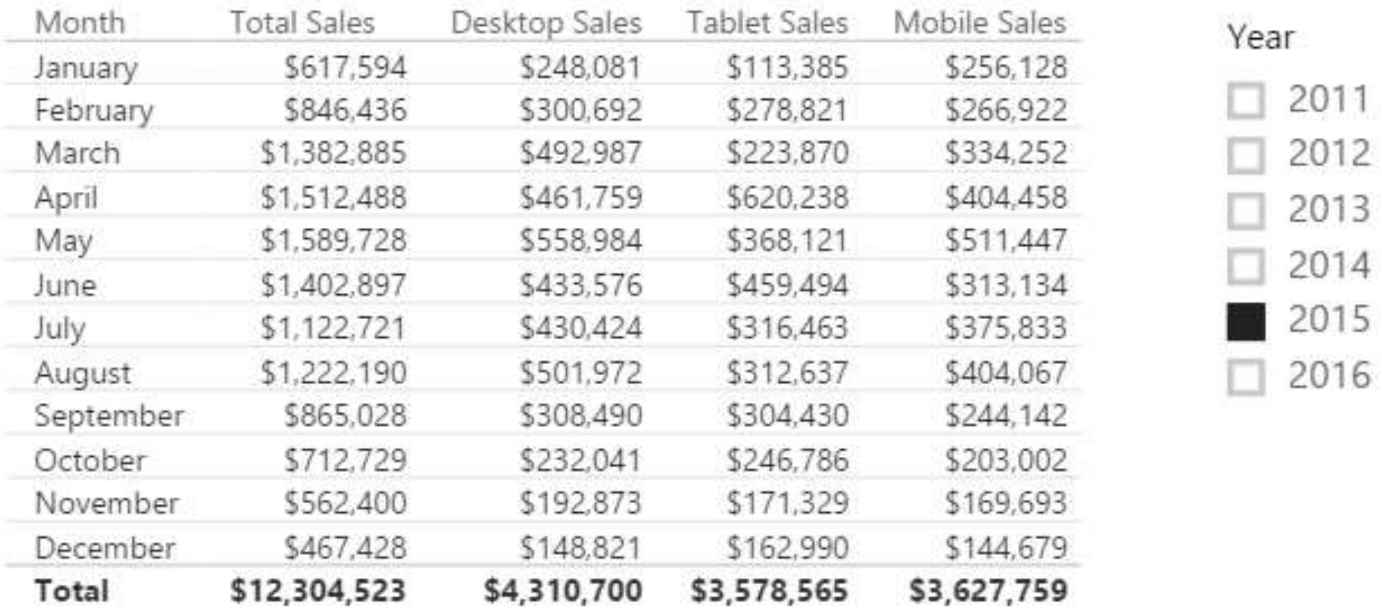
- Add Filter
- Ignore Filter
- Update Filter
- Convert Row Context to Filter Context

# CALCULATE – Add Filter

**[Desktop Sales] = CALCULATE([Total Sales], CampaignDim[Device] = "Desktop")**

**[Tablet Sales] = CALCULATE([Total Sales], CampaignDim[Device] = “Tablet")**

**[Mobile Sales] = CALCULATE([Total Sales], CampaignDim[Device] = “Mobile")**



| Month | Total Sales | Desktop Sales | Tablet Sales | Mobile Sales |
|---|---|---|---|---|
| January | $617,594 | $248,081 | $113,385 | $256,128 |
| February | $846,436 | $300,692 | $278,821 | $266,922 |
| March | $1,382,885 | $492,987 | $223,870 | $334,252 |
| April | $1,512,488 | $461,759 | $620,238 | $404,458 |
| May | $1,589,728 | $558,984 | $368,121 | $511,447 |
| June | $1,402,897 | $433,576 | $459,494 | $313,134 |
| July | $1,122,721 | $430,424 | $316,463 | $375,833 |
| August | $1,222,190 | $501,972 | $312,637 | $404,067 |
| September | $865,028 | $308,490 | $304,430 | $244,142 |
| October | $712,729 | $232,041 | $246,786 | $203,002 |
| November | $562,400 | $192,873 | $171,329 | $169,693 |
| December | $467,428 | $148,821 | $162,990 | $144,679 |
| **Total** | **$12,304,523** | **$4,310,700** | **$3,578,565** | **$3,627,759** |

Year

□ 2011
□ 2012
□ 2013
□ 2014
■ 2015
□ 2016

*When the Device Slicer is selected, only “Total Sales” changes.

# CALCULATE – Ignore Filter

## CALCULATE – The Most Important Function in DAX

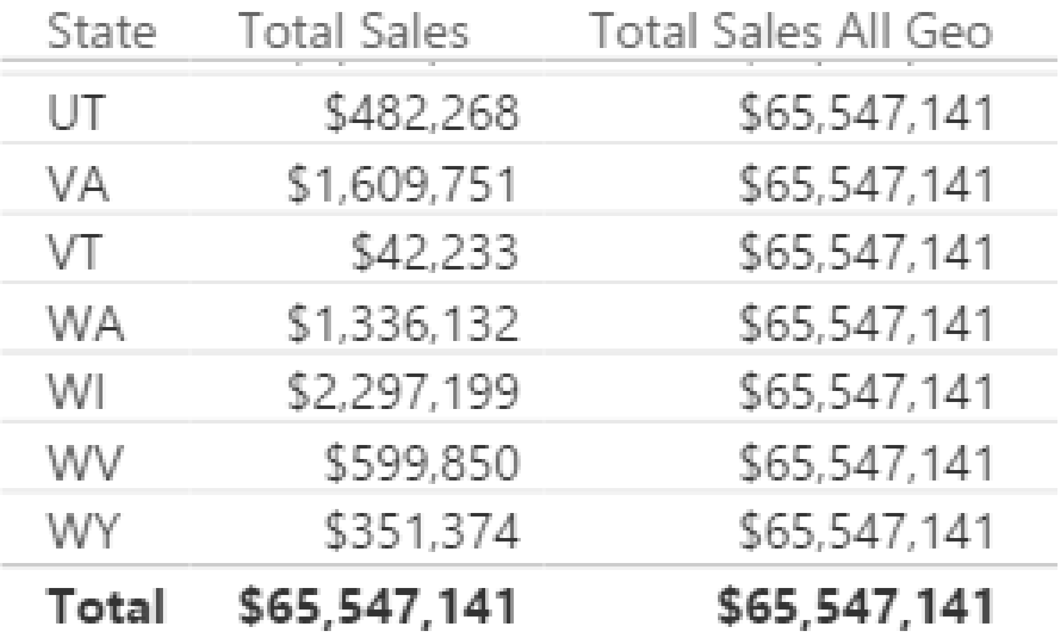
4 Key functions that CALCULATE can do

- Add Filter
- Ignore Filter
- Update Filter
- Convert Row Context to Filter Context

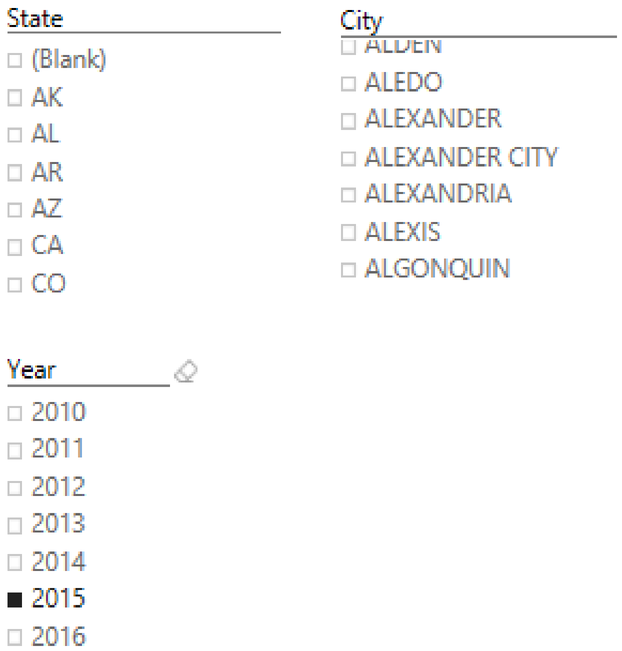
# CALCULATE – Ignore an Existing Filter

Power BI

**[Total Sales All Geo] = CALCULATE([Total Sales], ALL(GeographyDim))**



| State | Total Sales | Total Sales All Geo |
|---|---:|---:|
| UT | $482,268 | $65,547,141 |
| VA | $1,609,751 | $65,547,141 |
| VT | $42,233 | $65,547,141 |
| WA | $1,336,132 | $65,547,141 |
| WI | $2,297,199 | $65,547,141 |
| WV | $599,850 | $65,547,141 |
| WY | $351,374 | $65,547,141 |
| **Total** | **$65,547,141** | **$65,547,141** |



State

□ (Blank)
□ AK
□ AL
□ AR
□ AZ
□ CA
□ CO

City

□ ALDEN
□ ALEDO
□ ALEXANDER
□ ALEXANDER CITY
□ ALEXANDRIA
□ ALEXIS
□ ALGONQUIN

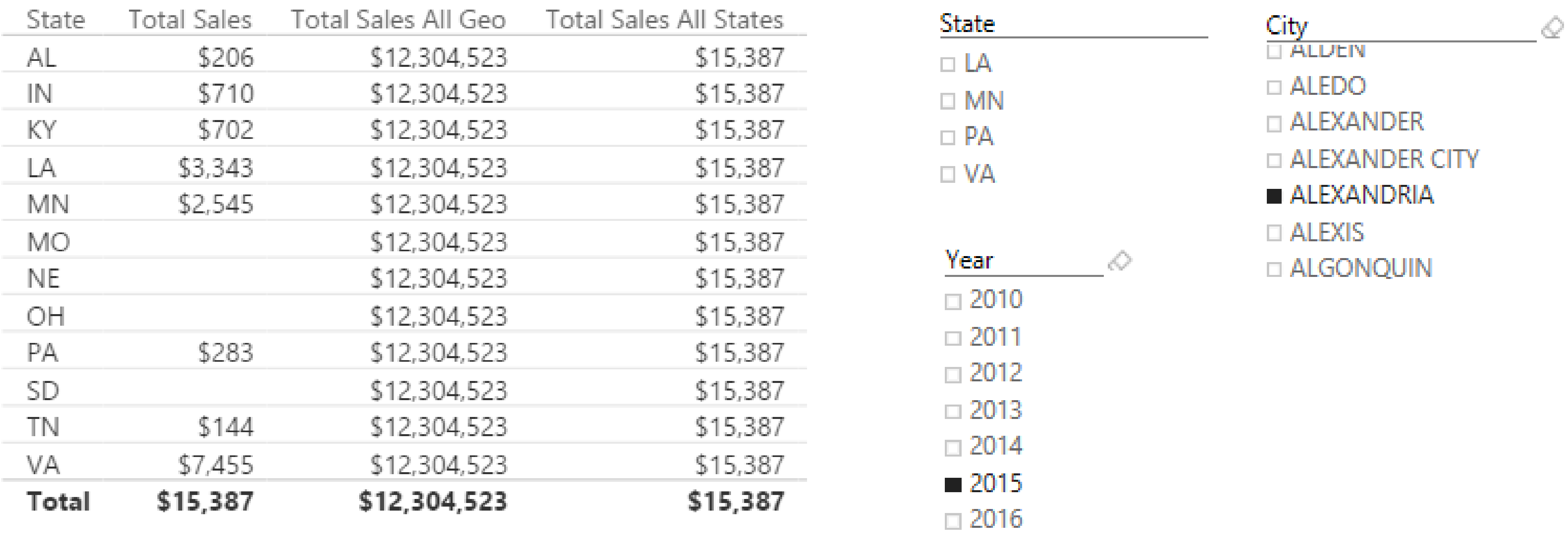
Year

□ 2010
□ 2011
□ 2012
□ 2013
□ 2014
■ 2015
□ 2016

*Ignore filter on ANY column from the GeographyDim table, but allows filters from Year

# CALCULATE – Ignore an Existing Filter

**[Total Sales All States] = CALCULATE([Total Sales], ALL(GeographyDim[State]))**



| State | Total Sales | Total Sales All Geo | Total Sales All States |
|---|---|---|---|
| AL | $206 | $12,304,523 | $15,387 |
| IN | $710 | $12,304,523 | $15,387 |
| KY | $702 | $12,304,523 | $15,387 |
| LA | $3,343 | $12,304,523 | $15,387 |
| MN | $2,545 | $12,304,523 | $15,387 |
| MO |  | $12,304,523 | $15,387 |
| NE |  | $12,304,523 | $15,387 |
| OH |  | $12,304,523 | $15,387 |
| PA | $283 | $12,304,523 | $15,387 |
| SD |  | $12,304,523 | $15,387 |
| TN | $144 | $12,304,523 | $15,387 |
| VA | $7,455 | $12,304,523 | $15,387 |
| **Total** | **$15,387** | **$12,304,523** | **$15,387** |

State

□ LA
□ MN
□ PA
□ VA

Year

□ 2010
□ 2011
□ 2012
□ 2013
□ 2014
■ 2015
□ 2016

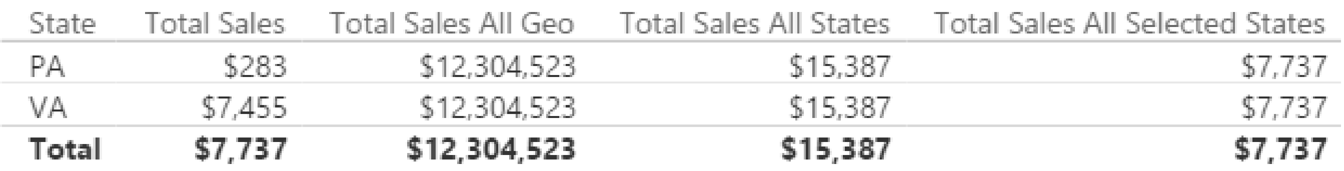
City

□ ALDEN
□ ALEDO
□ ALEXANDER
□ ALEXANDER CITY
■ ALEXANDRIA
□ ALEXIS
□ ALGONQUIN

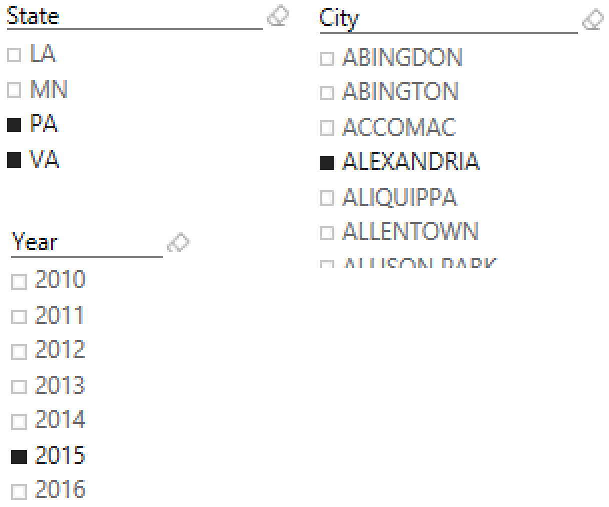
*Ignore filter on the STATE column from the GeographyDim table, but allows filters from Year

# CALCULATE – Ignore Existing Filter

**[Total Sales All Selected States] = CALCULATE([Total Sales], ALLSELECTED(GeographyDim[State]))**



| State | Total Sales | Total Sales All Geo | Total Sales All States | Total Sales All Selected States |
|---|---:|---:|---:|---:|
| PA | $283 | $12,304,523 | $15,387 | $7,737 |
| VA | $7,455 | $12,304,523 | $15,387 | $7,737 |
| **Total** | **$7,737** | **$12,304,523** | **$15,387** | **$7,737** |



State

□ LA
□ MN
■ PA
■ VA

Year

□ 2010
□ 2011
□ 2012
□ 2013
□ 2014
■ 2015
□ 2016

City

□ ABINGDON
□ ABINGTON
□ ACCOMAC
■ ALEXANDRIA
□ ALIQUIPPA
□ ALLENTOWN
□ ALLISON PARK

*Ignore filter on the STATE column from the GeographyDim table, but allows filters from Year

# CALCULATE – Update Filter

Power BI

## CALCULATE – The Most Important Function in DAX

4 Key functions that CALCULATE can do

Add Filter

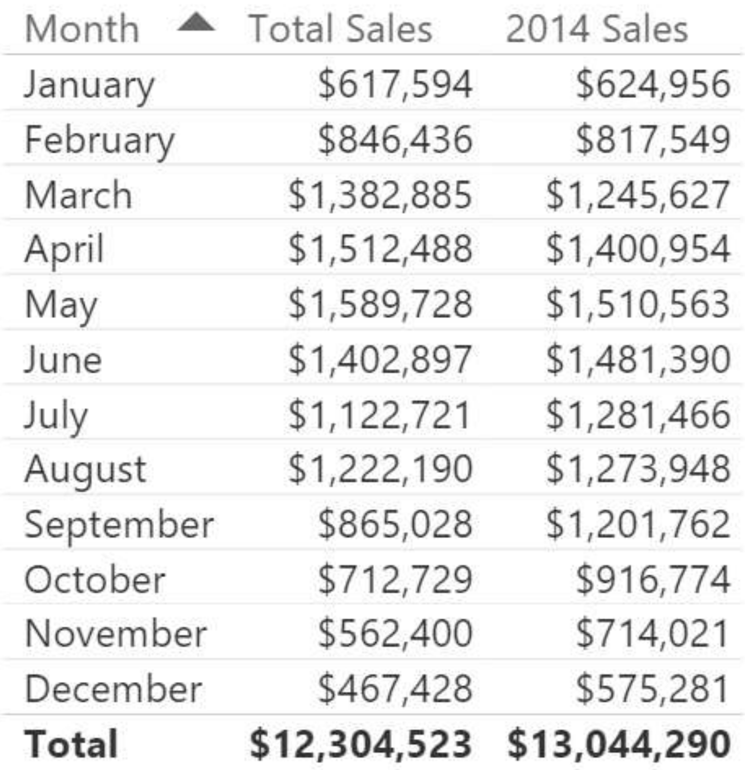
Ignore Filter

Update Filter

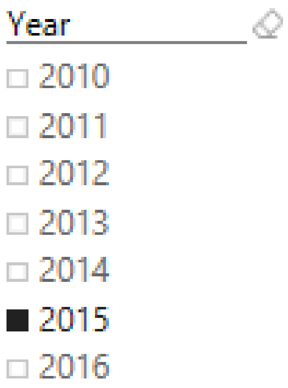
Convert Row Context to Filter Context

# CALCULATE – Update Existing Filter

**[2014 Sales] = CALCULATE([Total Sales], DateDim[Year] = 2014)**



| Month ▲ | Total Sales | 2014 Sales |
|---|---:|---:|
| January | $617,594 | $624,956 |
| February | $846,436 | $817,549 |
| March | $1,382,885 | $1,245,627 |
| April | $1,512,488 | $1,400,954 |
| May | $1,589,728 | $1,510,563 |
| June | $1,402,897 | $1,481,390 |
| July | $1,122,721 | $1,281,466 |
| August | $1,222,190 | $1,273,948 |
| September | $865,028 | $1,201,762 |
| October | $712,729 | $916,774 |
| November | $562,400 | $714,021 |
| December | $467,428 | $575,281 |
| **Total** | **$12,304,523** | **$13,044,290** |



Year

□ 2010
□ 2011
□ 2012
□ 2013
□ 2014
■ 2015
□ 2016

*Ignores filter on the Year Slicer

# CALCULATE – Convert Row Context to Filter Context

## CALCULATE – The Most Important Function in DAX

4 Key functions that CALCULATE can do

- Add Filter
- Ignore Filter
- Update Filter
- Convert Row Context to Filter Context

Let’s investigate what we mean by **Filter Context**

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Lunch

Use the time to get in contact with your classmates and instructors!!

# Agenda

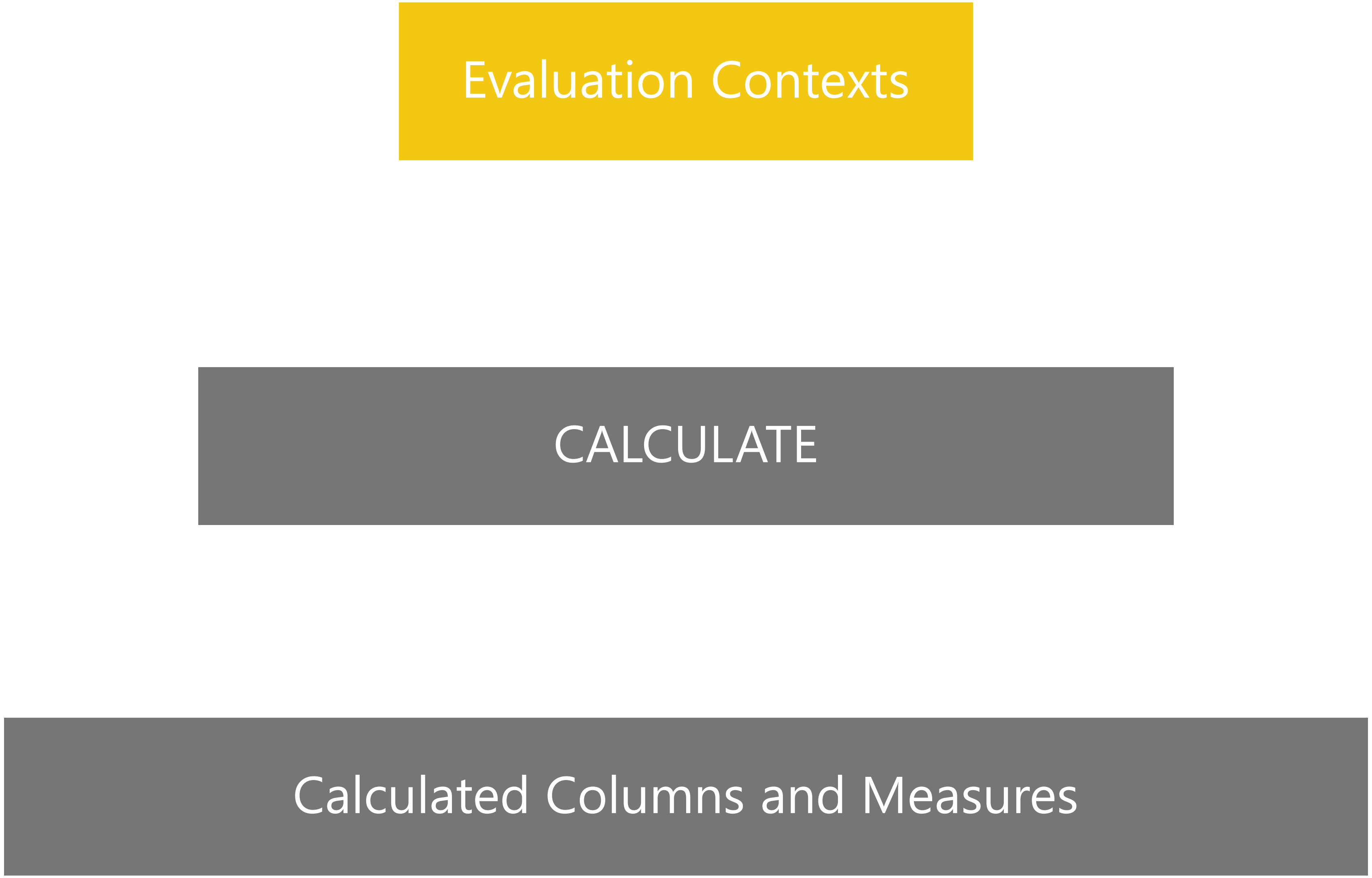
| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Module
# DAX Evaluation Contexts

# MODULE 4 OBJECTIVES

- Understand that there are different kinds of evaluation contexts and be able to explain what different contexts are in play
- Be able to use iterator functions and CALCULATE to create sophisticated measures

## PATH to DAX Expertise



Evaluation Contexts

CALCULATE

Calculated Columns and Measures

# Evaluation Context

Power BI

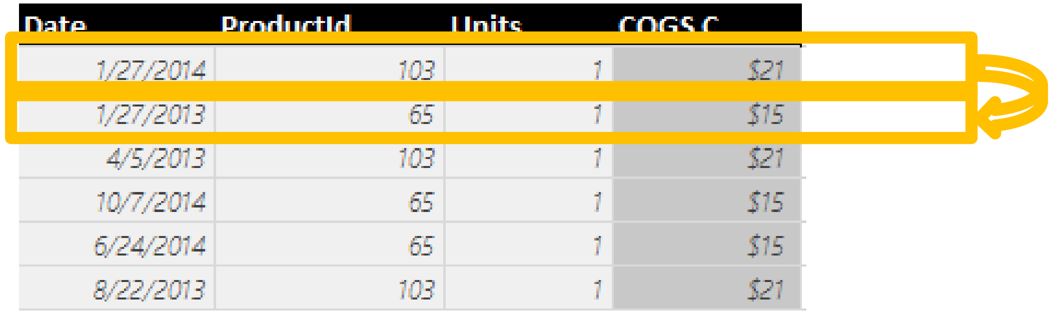
## There are two contexts under which calculations are evaluated

Row Context

Filter Context

# Row context in Calculated Column

**Sales[COGS] = RELATED(ProductDim[Unit Cost]) * Sales[Units]**



| Date | ProductId | Units | COGS C |
|---|---|---|---|
| 1/27/2014 | 103 | 1 | $21 |
| 1/27/2013 | 65 | 1 | $15 |
| 4/5/2013 | 103 | 1 | $21 |
| 10/7/2014 | 65 | 1 | $15 |
| 6/24/2014 | 65 | 1 | $15 |
| 8/22/2013 | 103 | 1 | $21 |

- Formula is evaluated row by row
- The context under which formula is evaluated for each row is called “Row Context”

**Pro Tip:** To accumulate up from Fact to Dimension, use **RELATEDTABLE()**

# Evaluation Context

**Both Calculated Columns and Measures are always evaluated under two contexts**

Row Context

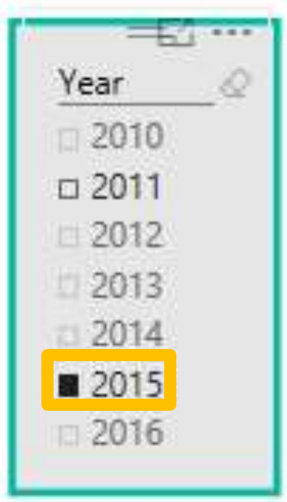
Filter Context

# Filter Context in Measures

## Filter Context in a Measure – Example 1

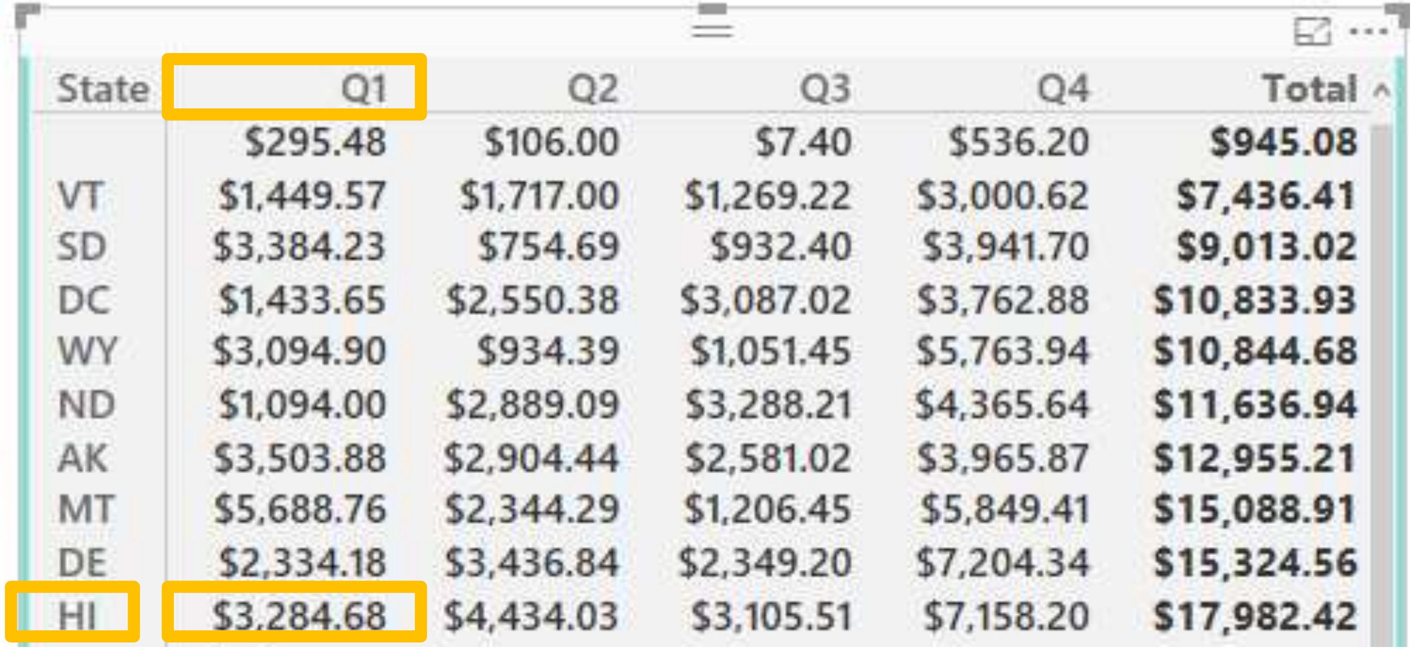
**[Total Sales] = SUM(Sales[Sales Amount])**

Filter Context for current coordinate Year = 2015, State = **HI**, Quarter = **Q1**



Year

- □ 2010
- □ 2011
- □ 2012
- □ 2013
- □ 2014
- ■ 2015
- □ 2016



| State | Q1 | Q2 | Q3 | Q4 | Total |
|---|---|---|---|---|---|
| | $295.48 | $106.00 | $7.40 | $536.20 | **$945.08** |
| VT | $1,449.57 | $1,717.00 | $1,269.22 | $3,000.62 | **$7,436.41** |
| SD | $3,384.23 | $754.69 | $932.40 | $3,941.70 | **$9,013.02** |
| DC | $1,433.65 | $2,550.38 | $3,087.02 | $3,762.88 | **$10,833.93** |
| WY | $3,094.90 | $934.39 | $1,051.45 | $5,763.94 | **$10,844.68** |
| ND | $1,094.00 | $2,889.09 | $3,288.21 | $4,365.64 | **$11,636.94** |
| AK | $3,503.88 | $2,904.44 | $2,581.02 | $3,965.87 | **$12,955.21** |
| MT | $5,688.76 | $2,344.29 | $1,206.45 | $5,849.41 | **$15,088.91** |
| DE | $2,334.18 | $3,436.84 | $2,349.20 | $7,204.34 | **$15,324.56** |
| HI | $3,284.68 | $4,434.03 | $3,105.51 | $7,158.20 | **$17,982.42** |

- Formula is evaluated for each "Coordinate" in each visual
- The context for each coordinate is called "Filter Context"

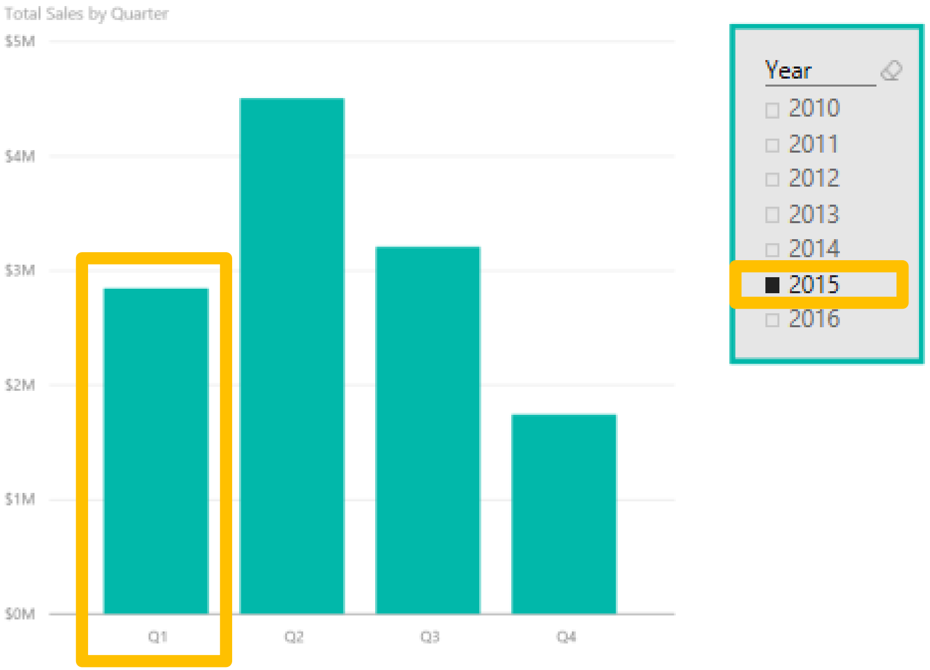
# Filter Context in a Measure

Power BI

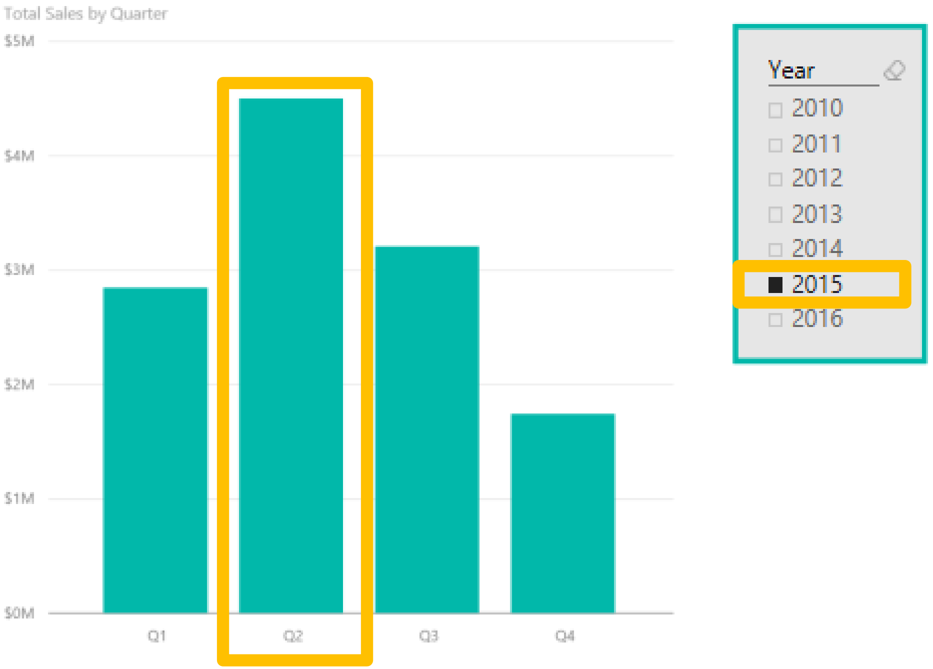
## Filter Context in a Measure – Example 2

**[Total Sales] = SUM(Sales[Sales Amount])**

Filter Context : Year = 2015, Quarter = **Q1**



Filter Context : Year = 2015, Quarter = **Q2**



# Filter Context in a Measure

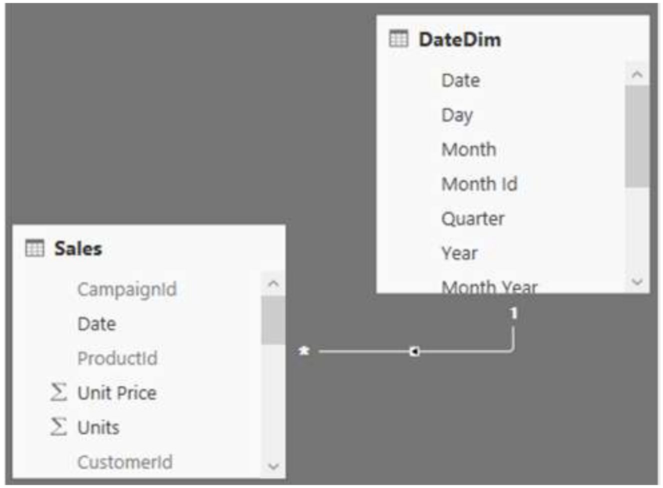
## Filter Context in a Measure

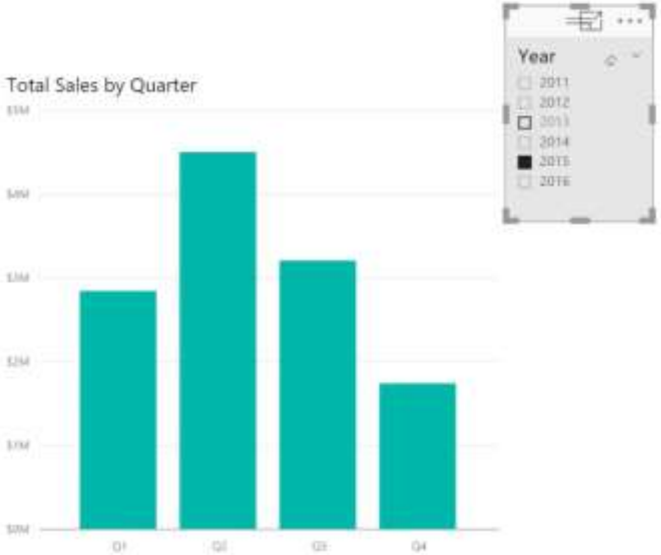
**[Total Sales] = SUM(Sales[Sales Amount])**

**Better definition of above measure:**

“Total Sales” – SUM of Sales[Sales Amount] column **under a filter context**

# Filter Context and Multiple Tables



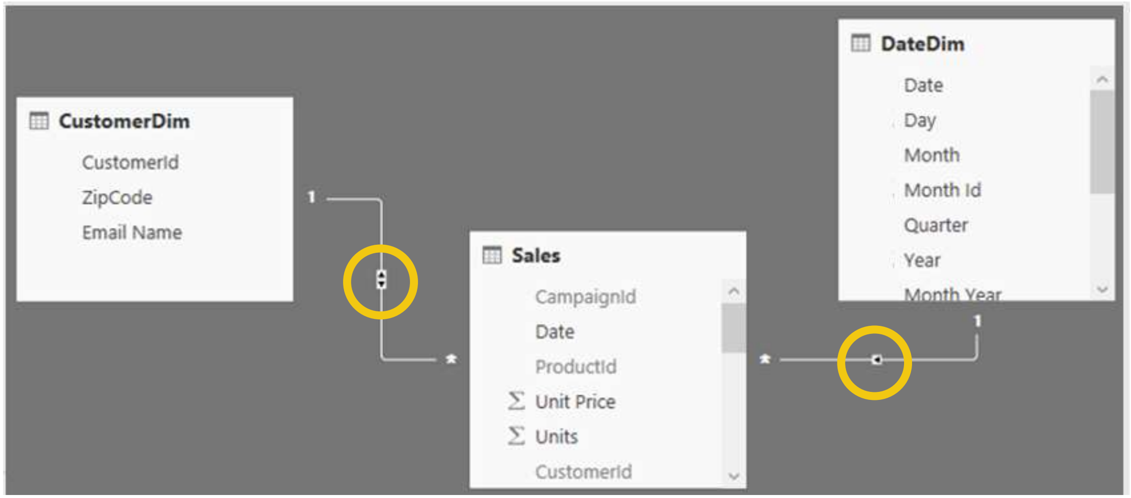


- Filter context automatically propagates from Dim Table to Fact Table
- Filtering the DateDim Table to Year = 2015 returns only Sales for 2015

# Filter Context and Multiple Tables

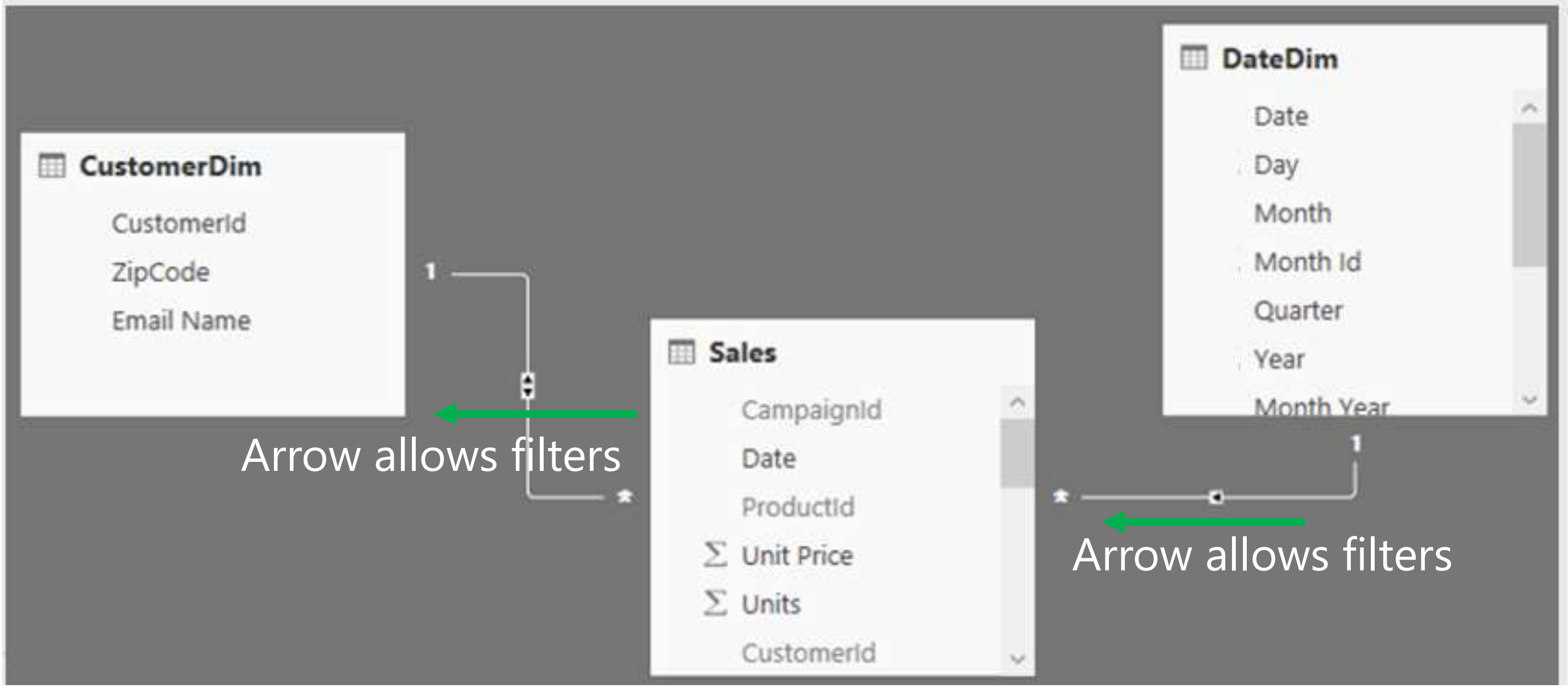
Power BI

## Filter Context and Multiple Tables



- Filters (Filter context) automatically propagate based on direction of arrows in relationships
- Examples
  - Filter goes from DateDim to CustomerDim
  - Filter does not go from CustomerDim to DateDim

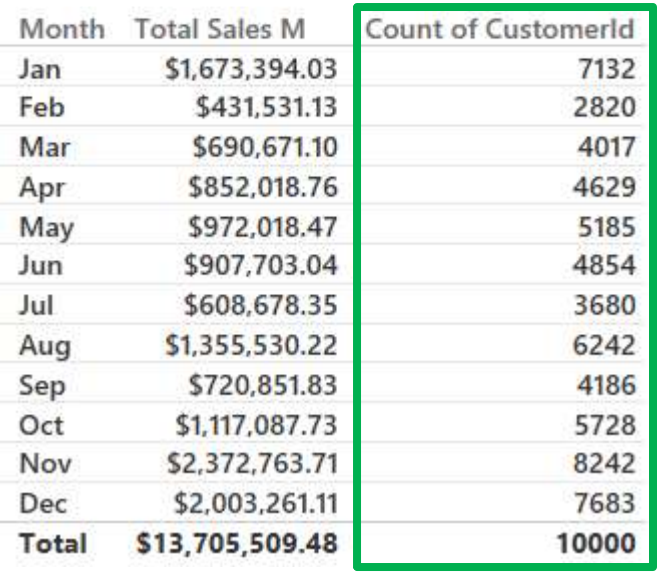
## Filter Context and Multiple Tables – Right Arrow Direction



Arrow allows filters

Arrow allows filters

**Cross filtering works properly**



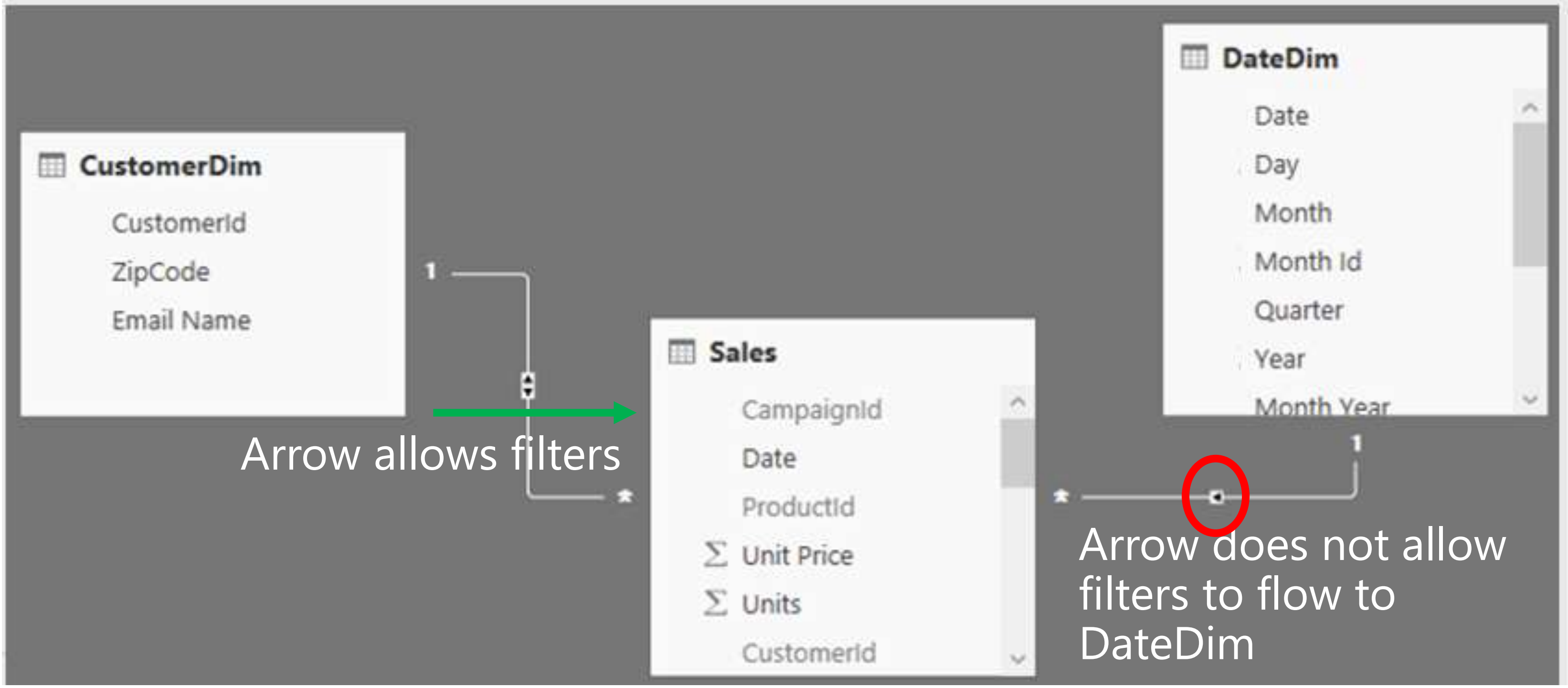
| Month | Total Sales M | Count of CustomerId |
|---|---:|---:|
| Jan | $1,673,394.03 | 7132 |
| Feb | $431,531.13 | 2820 |
| Mar | $690,671.10 | 4017 |
| Apr | $852,018.76 | 4629 |
| May | $972,018.47 | 5185 |
| Jun | $907,703.04 | 4854 |
| Jul | $608,678.35 | 3680 |
| Aug | $1,355,530.22 | 6242 |
| Sep | $720,851.83 | 4186 |
| Oct | $1,117,087.73 | 5728 |
| Nov | $2,372,763.71 | 8242 |
| Dec | $2,003,261.11 | 7683 |
| **Total** | **$13,705,509.48** | **10000** |

- Filter goes from DateDim to CustomerDim
- This is why the above Pivot table works

# Filter Context and Multiple Tables

Power BI

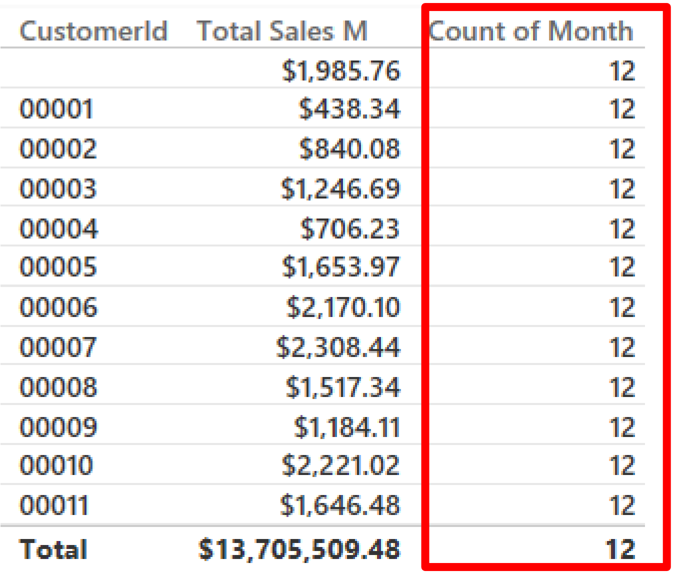
## Filter Context and Multiple Tables – Wrong Arrow Direction



Arrow allows filters

Arrow does not allow filters to flow to DateDim

Cross filtering does not work



| CustomerId | Total Sales M | Count of Month |
|---|---|---|
| | $1,985.76 | 12 |
| 00001 | $438.34 | 12 |
| 00002 | $840.08 | 12 |
| 00003 | $1,246.69 | 12 |
| 00004 | $706.23 | 12 |
| 00005 | $1,653.97 | 12 |
| 00006 | $2,170.10 | 12 |
| 00007 | $2,308.44 | 12 |
| 00008 | $1,517.34 | 12 |
| 00009 | $1,184.11 | 12 |
| 00010 | $2,221.02 | 12 |
| 00011 | $1,646.48 | 12 |
| **Total** | **$13,705,509.48** | **12** |

- Filter goes from DateDim to CustomerDim
- This is why the Count of Month in the table above is incorrect

# Bidirectional filters are dangerous!

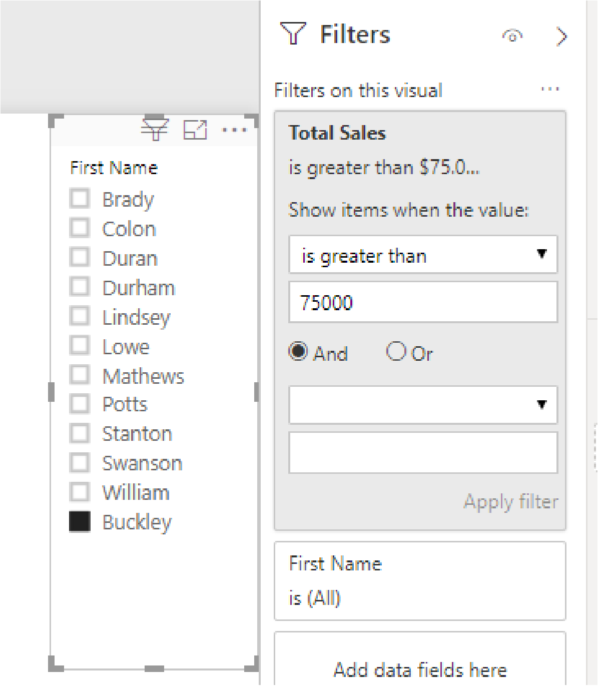
The presence of that bidirectional cross-filter is going to quickly create our worst nightmare

https://www.sqlbi.com/articles/bidirectional-relationships-and-ambiguity-in-dax/

WRITTEN BY

Alberto Ferrari

- Since June 2019 Power BI allows to filters slicers by a measure and thus many use cases for bi-directional relationships have gone away



First Name

□ Brady
□ Colon
□ Duran
□ Durham
□ Lindsey
□ Lowe
□ Mathews
□ Potts
□ Stanton
□ Swanson
□ William
■ Buckley

Filters

Filters on this visual

**Total Sales**

is greater than $75.0...

Show items when the value:

is greater than

75000

● And ○ Or

Apply filter

First Name

is (All)

Add data fields here

# Evaluation Context and Multiple Tables

## Evaluation Context Multiple Table – Summary and Take Aways

### Row Context

- Does not propagate automatically
- Need to use

  **RELATED**

  **RELATEDTABLE**

### Filter Context

- Propagates automatically
- Depends on direction of arrow in relationship diagram

# DAX Function Types

## Scalar Functions

- Scalar functions return a Single value as an output
- Ex. SUM(Sale[Sales Amount])

## Table Functions

- Table functions return a Table as an output
- Ex. ALL(GeographyDim)

There are other ways to classify functions – By kind of operation they perform etc.

# Applications of Table functions

Power BI

Table functions can be used 2 ways in Power BI Desktop

- As an input to another DAX function
  - CALCULATE
  - Iterator functions

- Calculated Tables

# Basic TABLE functions

**Return All Rows**

ALL &
variants

**Return Distinct Rows**

ALL, DISTINCT,
VALUES

**Return Filtered Rows**

FILTER

There are more advanced Table functions, which we will not cover

# Basic Table functions – Return All Rows

- The **ALL** function - Can take either Table or Columns in a Table as input

| **ALL with Entire Table** | **ALL with One Column** | **ALL with Multiple Columns** |
|---|---|---|
| **ALL(GeographyDim)** | **ALL(GeographyDim[Region]))** | **ALL(GeographyDim[Region],<br>GeographyDim[State])** |
| Returns all rows all columns in Table | Returns all unique values of Column | Returns all unique combinations of Column values |

# Basic Table functions – ALL versions

Power BI

- There are several forms of the ALL function
  - **ALL**
  - **ALLEXCEPT** - Return all columns in a Table except 1 or more columns
  - **ALLSELECTED** – Return all values in a column selected by users in Slicers
  - **ALLNONBLANKROW** – Return all non-Blank rows

# Basic Table Functions – Return Distinct Rows

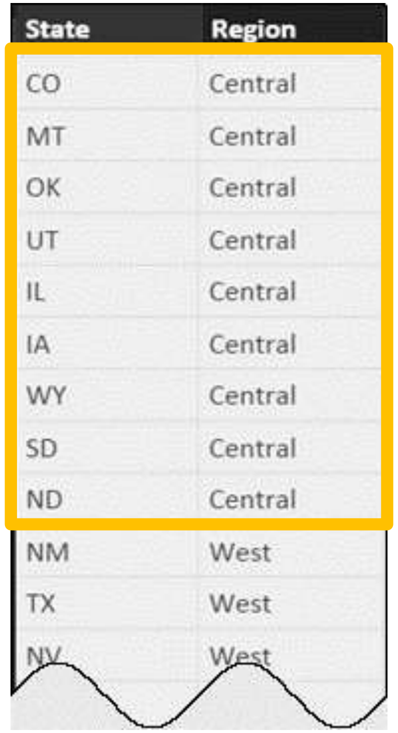
Power BI

- **VALUES –** Return all distinct values in a column or Table **(including blank rows)**
- **DISTINCT** - Return all distinct values in a column or Table **(not including blank rows)**

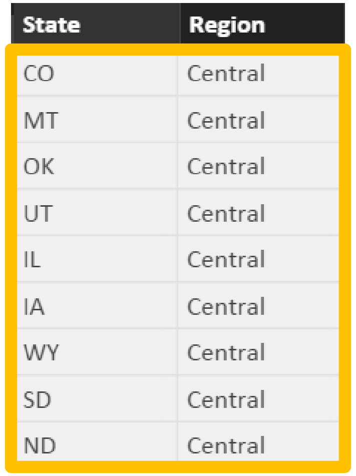
# Basic Table Functions – Return Filtered Set of Rows

**FILTER(ALL(GeographyDim[Region], GeographyDim[State]), GeographyDim[Region] = “Central”)**

- Take all unique combinations of GeographyDim[Region], GeographyDim[State]
- Filter down to the rows where GeographyDim[Region] = “Central”



| State | Region |
|---|---|
| CO | Central |
| MT | Central |
| OK | Central |
| UT | Central |
| IL | Central |
| IA | Central |
| WY | Central |
| SD | Central |
| ND | Central |
| NM | West |
| TX | West |
| NV | West |



| State | Region |
|---|---|
| CO | Central |
| MT | Central |
| OK | Central |
| UT | Central |
| IL | Central |
| IA | Central |
| WY | Central |
| SD | Central |
| ND | Central |

# DAX Iterator Functions

## DAX Iterator Functions Take Advantage of Evaluation Context

Iterator Functions

- Creates a *row context* by iterating over a table that you specify

- Ex. SUMX

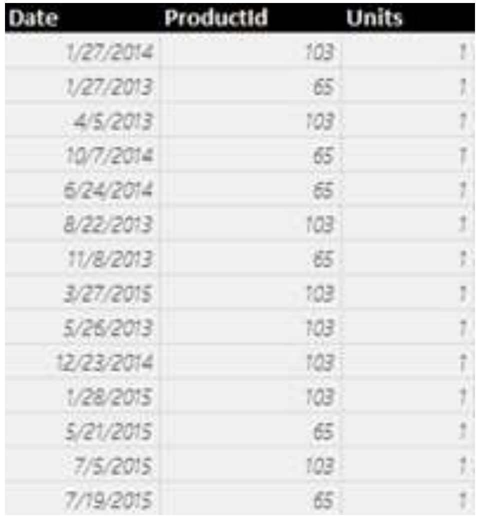
# Table Functions Application – Iterators

$$\text{[COGS] = SUMX(}\underbrace{\text{Sales,}}_{\textbf{Argument 1}}\ \underbrace{\text{Sales[Units] * RELATED(ProductDim[Unit Cost])}}_{\textbf{Argument 2}}\text{)}$$

# Table Functions Application – Iterators

[COGS] = SUMX(Sales, Sales[Units] * RELATED(ProductDim[Unit Cost]))

**Argument 1**



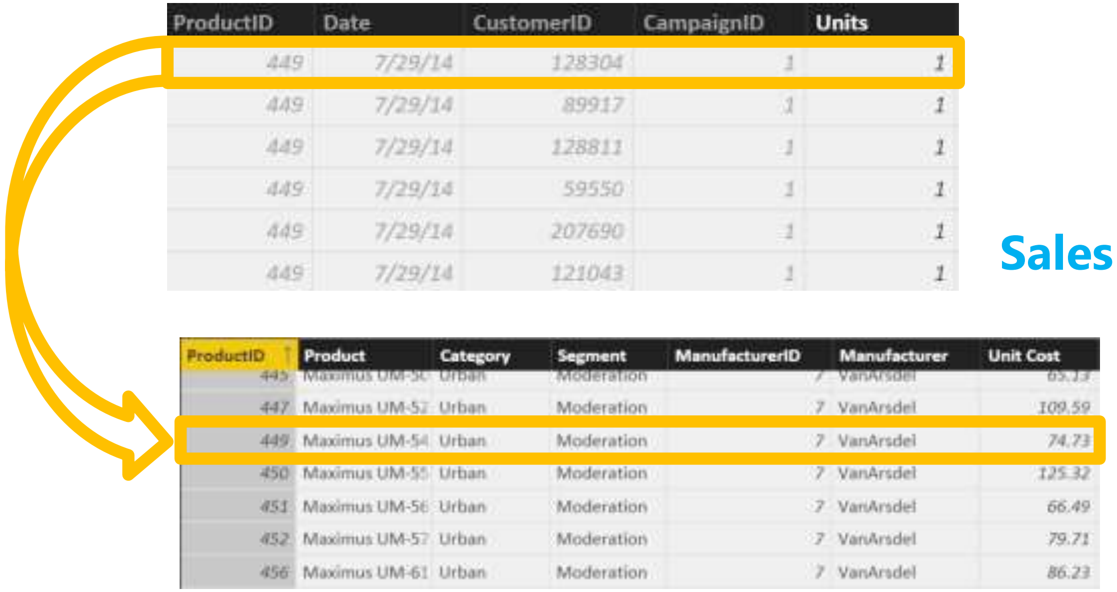
| Date | ProductId | Units |
|---|---|---|
| 1/27/2014 | 103 | 1 |
| 1/27/2013 | 65 | 1 |
| 4/5/2013 | 103 | 1 |
| 10/7/2014 | 65 | 1 |
| 6/24/2014 | 65 | 1 |
| 8/22/2013 | 103 | 1 |
| 11/8/2013 | 65 | 1 |
| 3/27/2015 | 103 | 1 |
| 5/26/2013 | 103 | 1 |
| 12/23/2014 | 103 | 1 |
| 1/28/2015 | 103 | 1 |
| 5/21/2015 | 65 | 1 |
| 7/5/2015 | 103 | 1 |
| 7/19/2015 | 65 | 1 |

**Iterate through each row in Argument 1**

**Sales**

# Table Functions Application – Iterators

$$\text{[COGS] = SUMX(Sales, }\underbrace{\text{Sales[Units] * RELATED(ProductDim[Unit Cost]))}}_{\text{Argument 2}}$$



| ProductID | Date | CustomerID | CampaignID | Units |
|---|---|---|---|---|
| 449 | 7/29/14 | 128304 | 1 | 1 |
| 449 | 7/29/14 | 89917 | 1 | 1 |
| 449 | 7/29/14 | 128811 | 1 | 1 |
| 449 | 7/29/14 | 59550 | 1 | 1 |
| 449 | 7/29/14 | 207690 | 1 | 1 |
| 449 | 7/29/14 | 121043 | 1 | 1 |

Sales

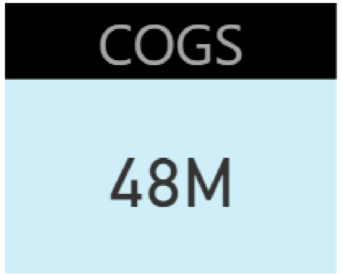
| ProductID | Product | Category | Segment | ManufacturerID | Manufacturer | Unit Cost |
|---|---|---|---|---|---|---|
| 445 | Maximus UM-5C | Urban | Moderation | 7 | VanArsdel | 65.13 |
| 447 | Maximus UM-52 | Urban | Moderation | 7 | VanArsdel | 109.59 |
| 449 | Maximus UM-54 | Urban | Moderation | 7 | VanArsdel | 74.73 |
| 450 | Maximus UM-55 | Urban | Moderation | 7 | VanArsdel | 125.32 |
| 451 | Maximus UM-56 | Urban | Moderation | 7 | VanArsdel | 66.49 |
| 452 | Maximus UM-57 | Urban | Moderation | 7 | VanArsdel | 79.71 |
| 456 | Maximus UM-61 | Urban | Moderation | 7 | VanArsdel | 86.23 |

ProductDim

# Row Context in a Measure – Iterator Functions

Power BI

$$\text{[COGS] = }\underbrace{\text{SUMX}}_{\text{SUM it up}}\text{(Sales, Sales[Units] * RELATED(ProductDim[Unit Cost]))}$$



COGS

48M

**SUM up list obtained**

# Iterators

## Why Can an Iterator be a Better Approach then a Calculated Column?

- You avoid creating a Calculated Column
- Let us see the impact of a Calculated Column Called COGS on Data model with 100K rows - What if we have 10 M rows?
- Iterators help you avoid several “Intermediate Calculated Columns”

## CALCULATE – Converting Row Context to Filter Context (Example 1)

```
Sales velocity Segment = IF(
                               SUMX(RELATEDTABLE(Sales), Sales[Sales Amount])>=200000,
                              “High  Velocity”,
                              “Low Velocity”)
```

```
Sales Velocity (Using CALCULATE) = IF (
                                          CALCULATE(SUM(Sales[Sales Amount])) >= 200000,
                                          "High Velocity",
                                          "Low Velocity")
```

- Another way to do Dynamic Segmentation
- This method does not use Iterators
- Instead it uses CALCULATE to convert Row Context to Filter Context

# Other Iterator Functions

Power BI

**AVERAGEX , PRODUCTX, MINX, MAXX– All work the same way as SUMX**

**RANKX – Works similar to SUMX, but slightly more complex (more options)**

# Table Functions – Summary and Application

Table functions can be used in 2 ways in Power BI Desktop:

- As an input to another DAX function
  - CALCULATE
  - Iterator functions

- Calculated Tables

**CALCULATE is one of the primary places where Table functions are used**

# KNOWLEDGE CHECK Module

- What are the different kinds of evaluation contexts?
- When are filter or a row contexts present?
- Which functions are commonly used to *modify* existing evaluation contexts?

# KNOWLEDGE CHECK ANSWERS Module

- What are the different kinds of evaluation contexts?
  - *Filter context and row context*

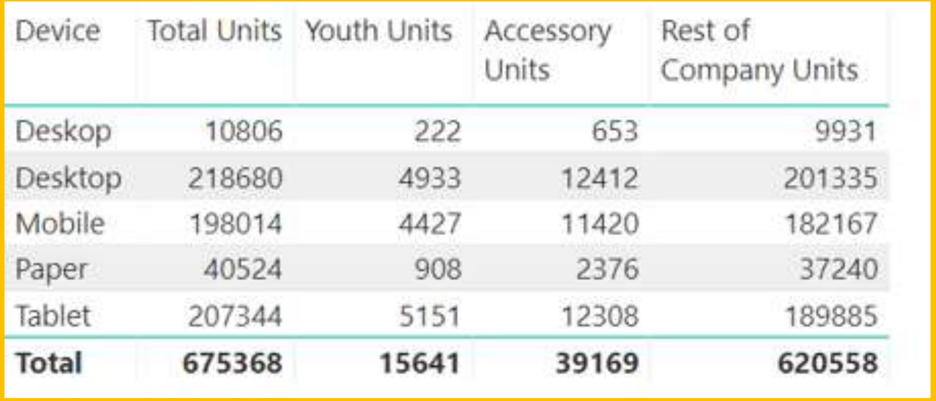
- When are filter or a row contexts present?
  - *Row contexts are present in iterator functions and calculated column evaluations. Filter contexts are present in pivot tables and other visualizations.*

- Which functions are commonly used to *modify* existing evaluation contexts?
  - *CALCULATE, ALL, etc.*

# Module Lab

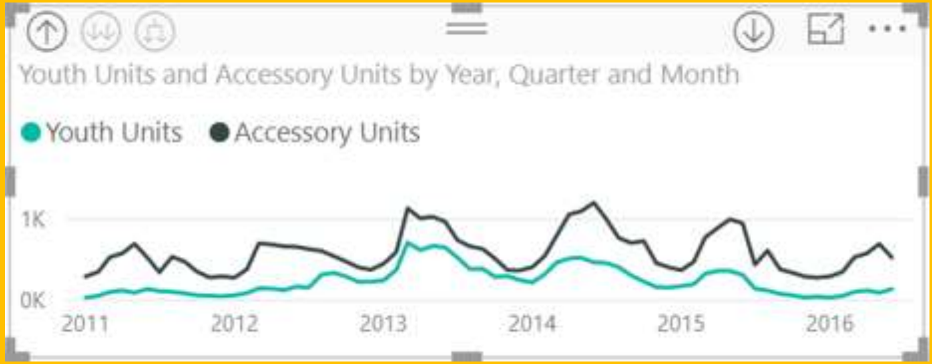
Create a report for the VP in charge of the Youth and Accessory Segments

1. Include a table visualization showing total units sold in the Youth Segment, Accessory Segment, and all other segments; by Campaign Device



| Device | Total Units | Youth Units | Accessory Units | Rest of Company Units |
|---|---|---|---|---|
| Deskop | 10806 | 222 | 653 | 9931 |
| Desktop | 218680 | 4933 | 12412 | 201335 |
| Mobile | 198014 | 4427 | 11420 | 182167 |
| Paper | 40524 | 908 | 2376 | 37240 |
| Tablet | 207344 | 5151 | 12308 | 189885 |
| **Total** | **675368** | **15641** | **39169** | **620558** |

2. Include a line chart showing total units sold in Youth and Accessory Segments by month



3. BONUS: Use the Unit Cost and Unit Price from the ProductDim table to calculate Sales Amount, Cost of Goods Sold, Profit and build some visuals around them

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Module

# Advanced DAX
Time Intelligence Functions

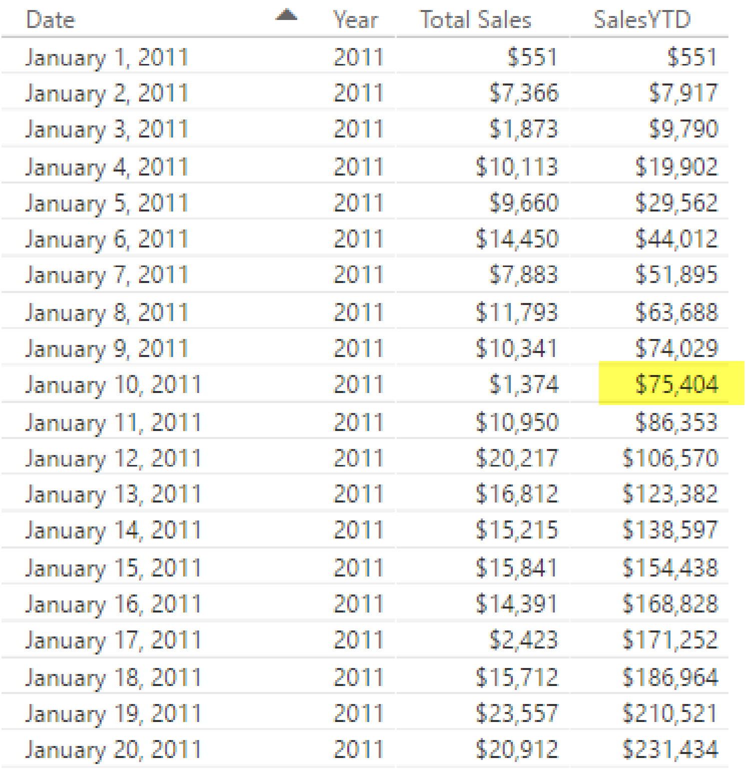
# MODULE OBJECTIVES

- Be able to parse advanced DAX formulas (e.g., cumulative functions)
- Gain familiarity with standard DAX patterns
- Introduction to resources for further learning

# Advanced DAX

## Before we get to Time Intelligence - Let us apply all of the DAX techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX(DateDim[Date])
    )
)
```



| Date | Year | Total Sales | SalesYTD |
|---|---|---|---|
| January 1, 2011 | 2011 | $551 | $551 |
| January 2, 2011 | 2011 | $7,366 | $7,917 |
| January 3, 2011 | 2011 | $1,873 | $9,790 |
| January 4, 2011 | 2011 | $10,113 | $19,902 |
| January 5, 2011 | 2011 | $9,660 | $29,562 |
| January 6, 2011 | 2011 | $14,450 | $44,012 |
| January 7, 2011 | 2011 | $7,883 | $51,895 |
| January 8, 2011 | 2011 | $11,793 | $63,688 |
| January 9, 2011 | 2011 | $10,341 | $74,029 |
| January 10, 2011 | 2011 | $1,374 | $75,404 |
| January 11, 2011 | 2011 | $10,950 | $86,353 |
| January 12, 2011 | 2011 | $20,217 | $106,570 |
| January 13, 2011 | 2011 | $16,812 | $123,382 |
| January 14, 2011 | 2011 | $15,215 | $138,597 |
| January 15, 2011 | 2011 | $15,841 | $154,438 |
| January 16, 2011 | 2011 | $14,391 | $168,828 |
| January 17, 2011 | 2011 | $2,423 | $171,252 |
| January 18, 2011 | 2011 | $15,712 | $186,964 |
| January 19, 2011 | 2011 | $23,557 | $210,521 |
| January 20, 2011 | 2011 | $20,912 | $231,434 |

Let us take a super complicated DAX statement and break it down and understand what it means

## Before we get to Time Intelligence - Let us apply all of the DAX techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX( DateDim[Date] )
    )
)
```

- In a CALCULATE statement Filter arguments are evaluated first

- The Filter in this case comes from a FILTER function

- FILTER function is an iterator

## Let us apply all of the data modeling techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX( DateDim[Date] )
    )
)
```

- In a FILTER statement the input Table is evaluated first
- ALL statement means take all DateDim

# Advanced DAX

## Let us apply all of the data modeling techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX(DateDim[Date] )
    )
)
```

- Iterate through each row in DateDim
- Check for condition based on row context and filter context

**Pro Tip:** if you need to concatenate two conditions with an **AND** use **&&** for and **OR** use **||**

# Advanced DAX

## Let us apply all of data modeling techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX(DateDim[Date] )
    )
)
```

- Now you have a FILTERED list of dates
- Use this to update the filter context (since it is in a CALCULATE statement)

## Let us apply all of the data modeling techniques

```
[SalesYTD] =

CALCULATE (
    [Total Sales],
    FILTER (
        ALL ( DateDim),
        DateDim[Year] = MAX ( DateDim[Year] )
            && DateDim[Date] <= MAX(DateDim[Date] )
    )
)
```

- Use updated FILTER context to evaluate ‘Total Sales’

# Advanced DAX – Time Intelligence

## Introducing Time Intelligence – There is a Function for that!!

```
[SalesYTD Easier] =

CALCULATE (
    [Total Sales],
    DATESYTD(DateDim[Date])
)
```

- This allows you to write the formula without being a DAX guru!
- Microsoft is continuously improving Time Intelligence functions to make it simple to use

**Time Intelligence functions are your friends – They will save you time!**

# Advanced DAX – Time Intelligence

```
[SalesYTD Even Easier] =

TOTALYTD(
    [Total Sales]
)
```

- The somewhat big DAX expression can be reduced to a single DAX expression

**Time Intelligence functions are your friends – They will save you time!**

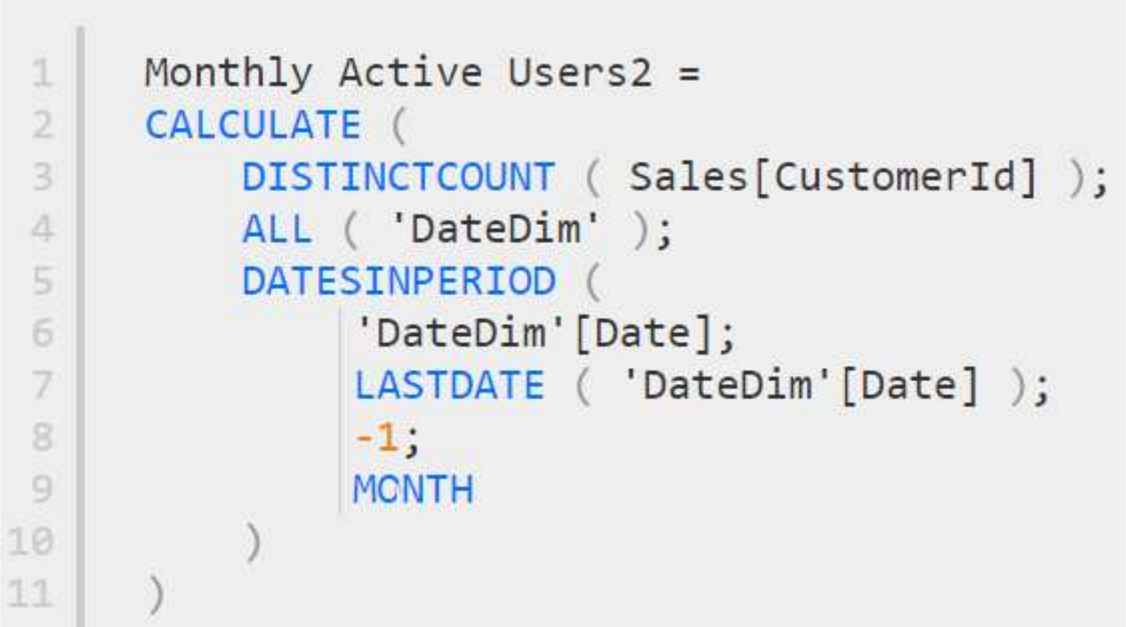
# Advanced DAX – Month over Month

Power BI

```
Total Sales Last Month =

CALCULATE([Total Sales],

    PREVIOUSMONTH(DateDim[Date]))



MoM =

DIVIDE([Total Sales] - [Total Sales Last Month],

   [Total Sales Last Month])
```

- DAX has several shortcut Time Intelligence functions

# Advanced DAX – Monthly Active Users



```
Monthly Active Users2 =
CALCULATE (
    DISTINCTCOUNT ( Sales[CustomerId] );
    ALL ( 'DateDim' );
    DATESINPERIOD (
        'DateDim'[Date];
        LASTDATE ( 'DateDim'[Date] );
        -1;
        MONTH
    )
)
```

- DATESINPERIOD allows running totals

# Advanced DAX – Time Intelligence

## Other Time Intelligence Functions

- DATESINPERIOD
- DATESYTD
- DATESQTD
- NEXTMONTH
- NEXTYEAR

- PREVIOUSYEAR
- PREVIOUSMONTH
- SAMEPERIODLASTYEAR
- PARALLELPERIOD

**Pro Tip: Learn about Time Intelligence functions - https://msdn.microsoft.com/en-us/library/ee634763.aspx**

# KNOWLEDGE CHECK Module

- Can I parse advanced DAX formulas?
- What are some standard DAX patterns?
- Which time intelligence functions are built-in to DAX?

# KNOWLEDGE CHECK ANSWERS Module

- Can I parse advanced DAX formulas?
  - *Yes I can!*

- What are some standard DAX patterns?
  - *CALCULATE(…)*

- Which time intelligence functions are built-in to DAX?
  - *Lots of them…YTD, FY, previous month, etc*

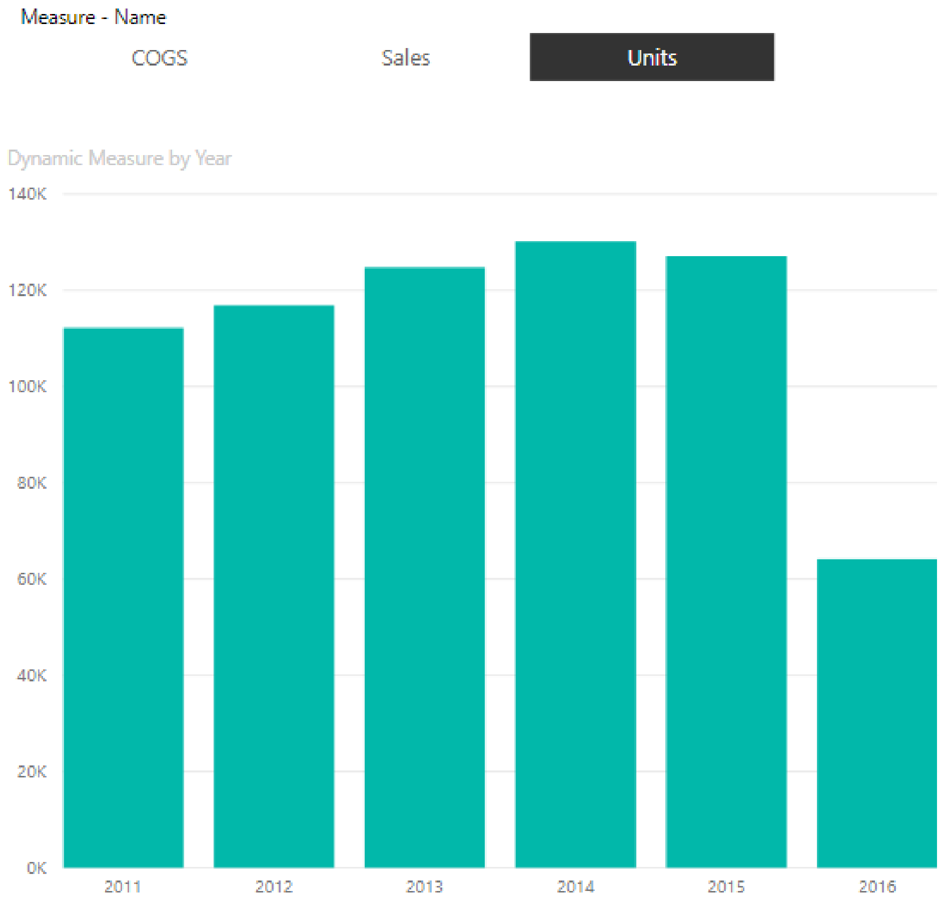
# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Measure Switching

# Measure Switching – Challenge

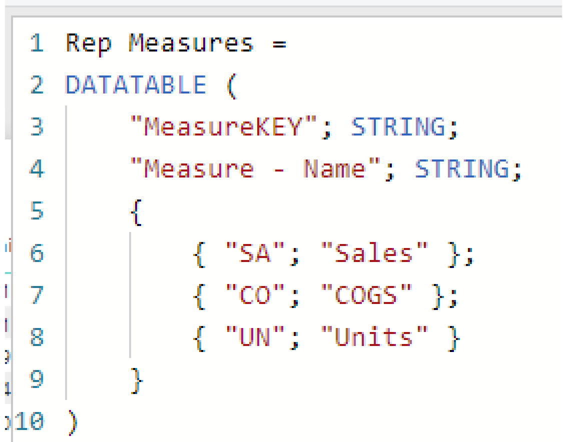
- A user wants to be able to influence on the fly what measures are shown in visuals while keeping the state of visuals
  - It helps you save space being able to switch rather than having many similar tables on a report
  - It increases interactivity
  - Bookmarks: will reset the state on clicking



# Measure Switching – Implementation I/

- We will create a slicer to use the values from Units, COGS and Sales in a graph

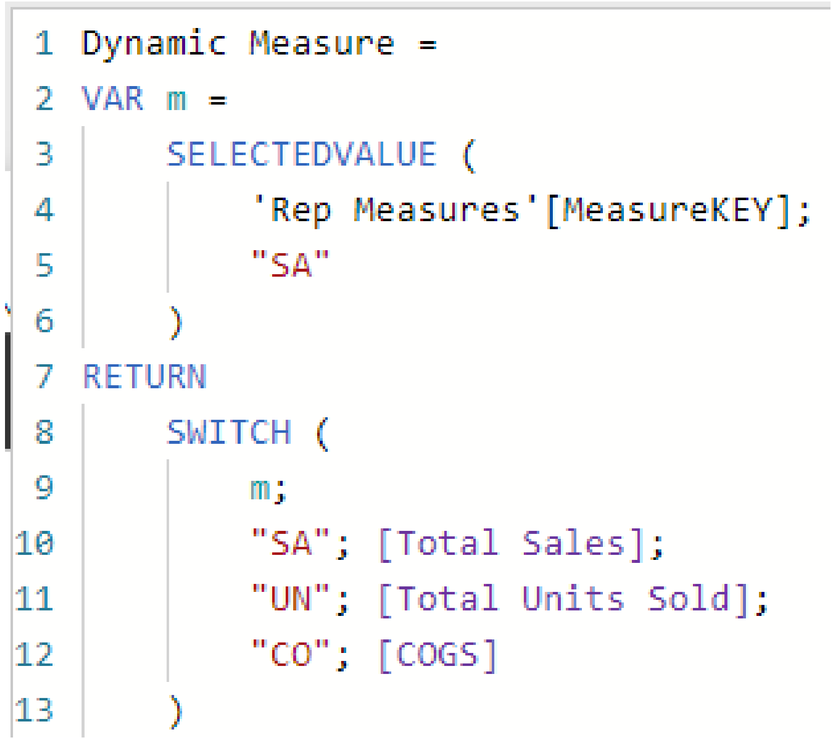
  1. We need to create a table with those names, remember also to name the table and the column that has the values



```
Rep Measures =
DATATABLE (
    "MeasureKEY"; STRING;
    "Measure - Name"; STRING;
    {
        { "SA"; "Sales" };
        { "CO"; "COGS" };
        { "UN"; "Units" }
    }
)
```

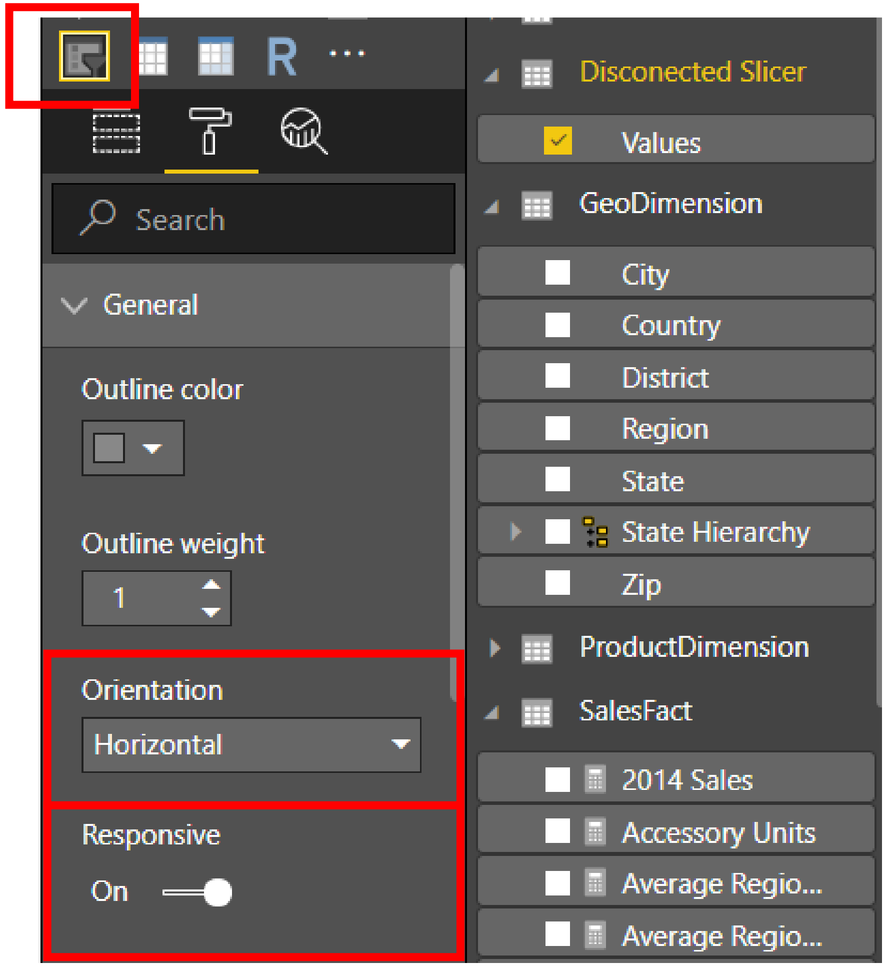
# Measure Switching – Implementation II/

2. Create a measure to link each values of the new table to the respective measures



```
Dynamic Measure =
VAR m =
    SELECTEDVALUE (
        'Rep Measures'[MeasureKEY];
        "SA"
    )
RETURN
    SWITCH (
        m;
        "SA"; [Total Sales];
        "UN"; [Total Units Sold];
        "CO"; [COGS]
    )
```

# Measure Switching – Implementation III/



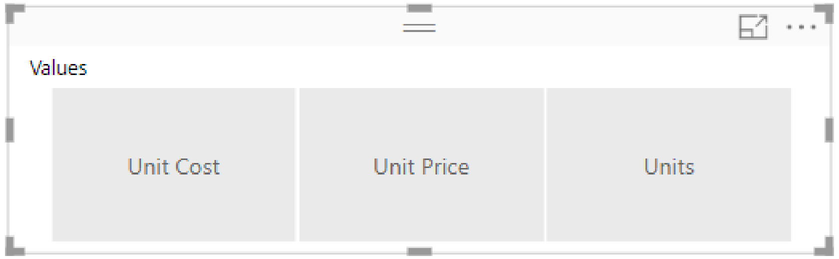
Outline color

Outline weight: 1

Orientation: Horizontal

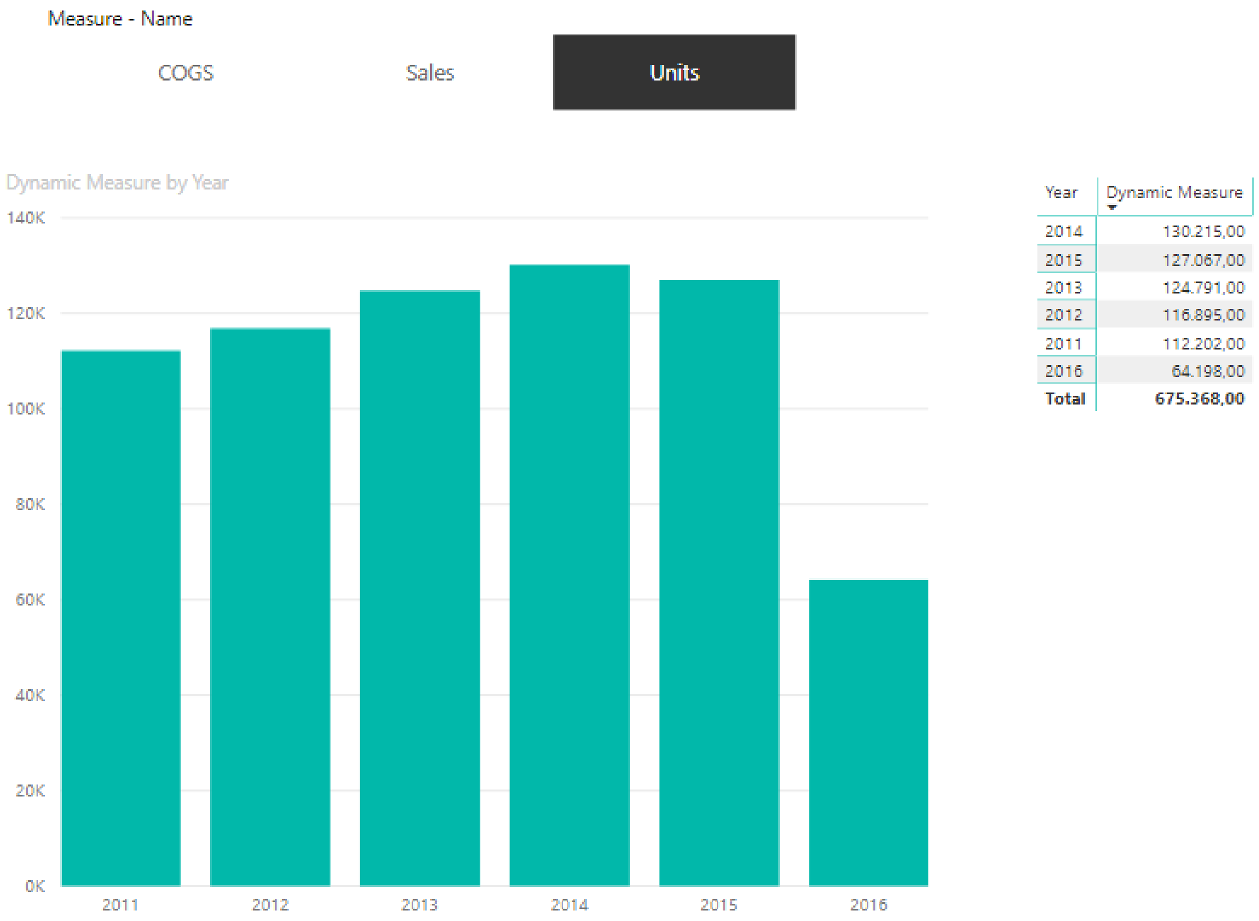
Responsive: On

3. Create a Slicer with the column of our new table



avanade

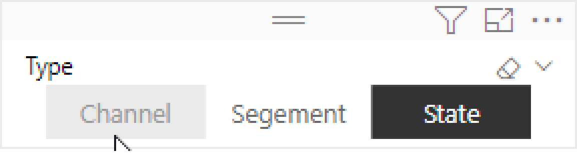
# Measure Switching – Result



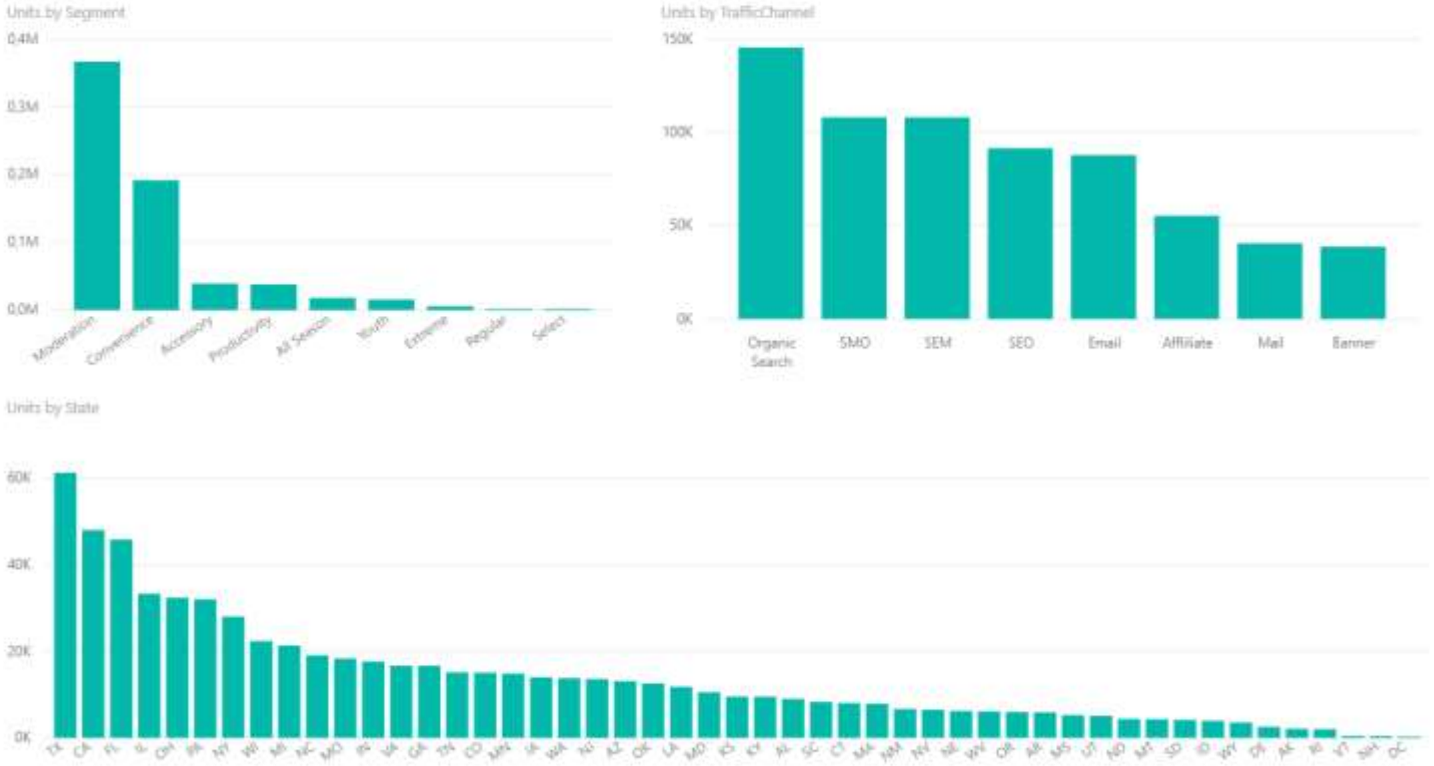
| Year | Dynamic Measure |
|---|---|
| 2014 | 130.215,00 |
| 2015 | 127.067,00 |
| 2013 | 124.791,00 |
| 2012 | 116.895,00 |
| 2011 | 112.202,00 |
| 2016 | 64.198,00 |
| **Total** | **675.368,00** |

4. Then create one or more visuals to apply the measure that we created

# Dimension Switching – Challenge



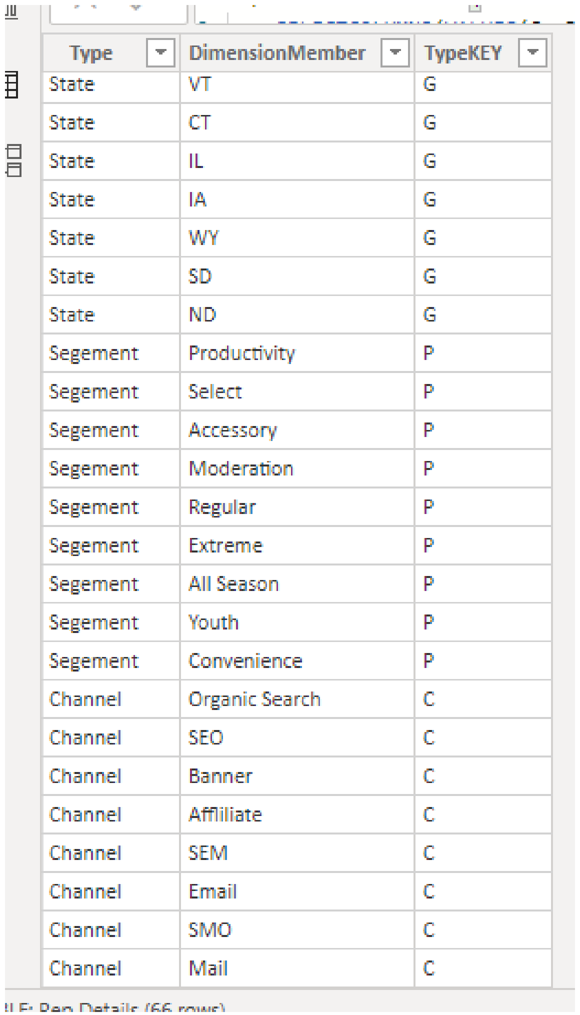
A user wants to have the option to dynamically put different dimensions on the axis and switch between them with a slicer



# Dimension Switching

# Dimension Switching – Implementation I/

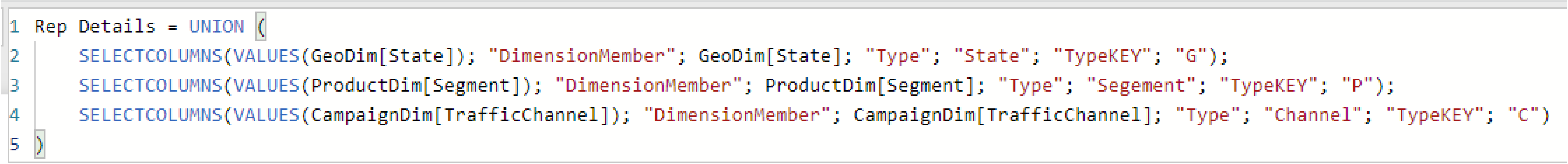
1. Create a table that contains the dimensions members and dimension names (and a key)



| Type | DimensionMember | TypeKEY |
|---|---|---|
| State | VT | G |
| State | CT | G |
| State | IL | G |
| State | IA | G |
| State | WY | G |
| State | SD | G |
| State | ND | G |
| Segement | Productivity | P |
| Segement | Select | P |
| Segement | Accessory | P |
| Segement | Moderation | P |
| Segement | Regular | P |
| Segement | Extreme | P |
| Segement | All Season | P |
| Segement | Youth | P |
| Segement | Convenience | P |
| Channel | Organic Search | C |
| Channel | SEO | C |
| Channel | Banner | C |
| Channel | Affiliate | C |
| Channel | SEM | C |
| Channel | Email | C |
| Channel | SMO | C |
| Channel | Mail | C |

LE: Rep Details (66 rows)

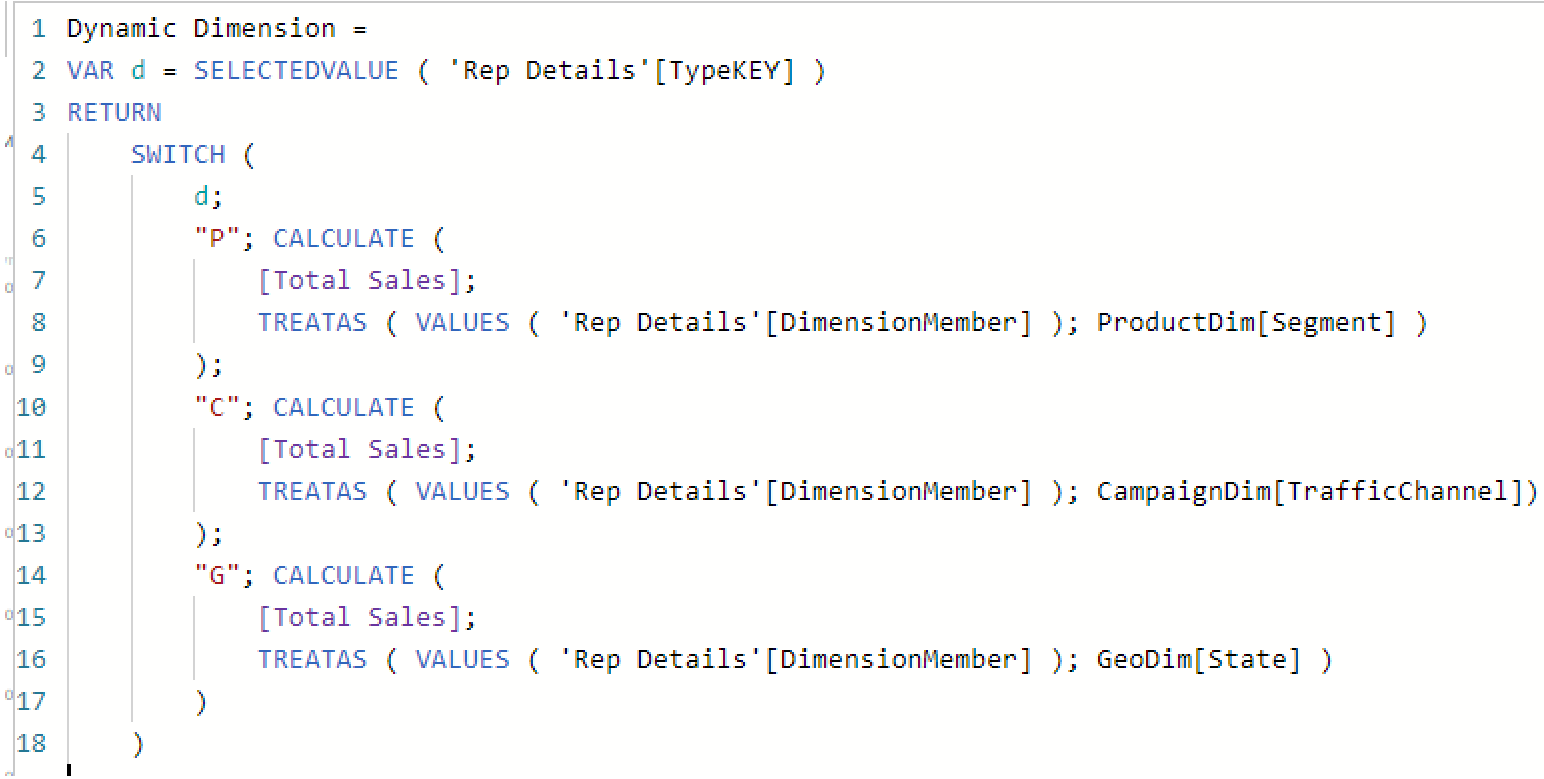
# Dimension Switching – Implementation II/



```
Rep Details = UNION (
    SELECTCOLUMNS(VALUES(GeoDim[State]); "DimensionMember"; GeoDim[State]; "Type"; "State"; "TypeKEY"; "G");
    SELECTCOLUMNS(VALUES(ProductDim[Segment]); "DimensionMember"; ProductDim[Segment]; "Type"; "Segement"; "TypeKEY"; "P");
    SELECTCOLUMNS(VALUES(CampaignDim[TrafficChannel]); "DimensionMember"; CampaignDim[TrafficChannel]; "Type"; "Channel"; "TypeKEY"; "C")
)
```

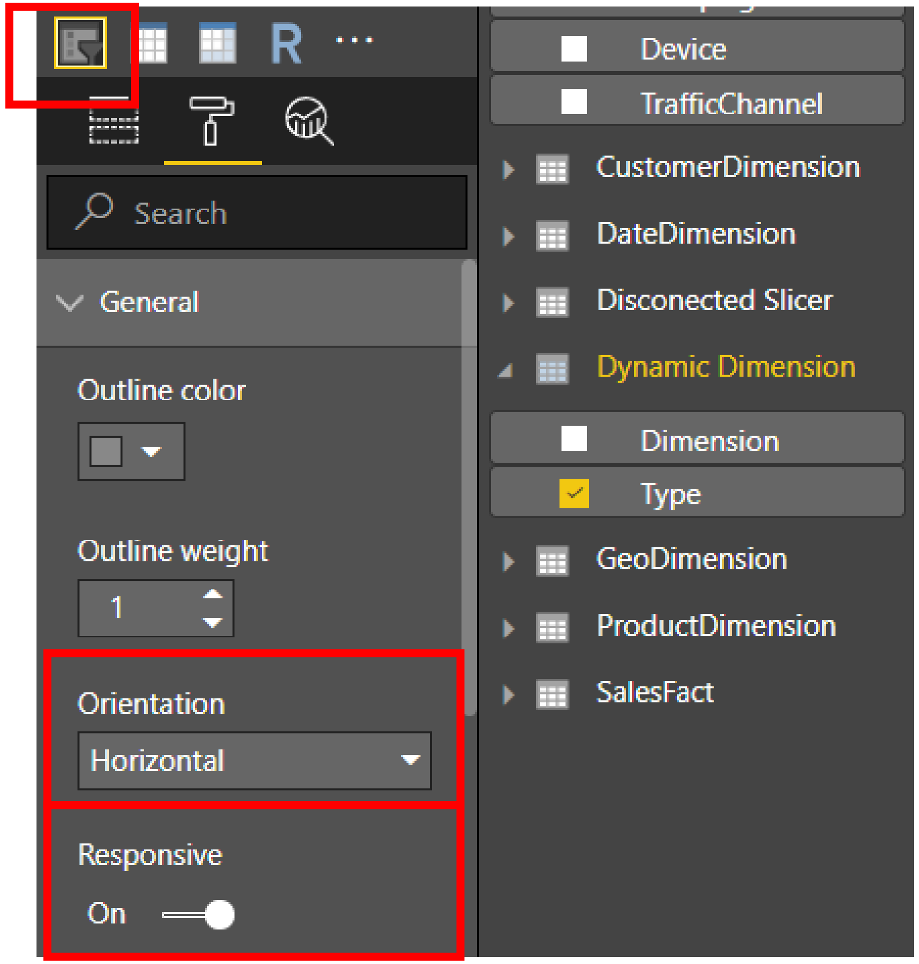
# Dimension Switching – Implementation III/

2. Create a measure to link it to the table and use it in the visuals

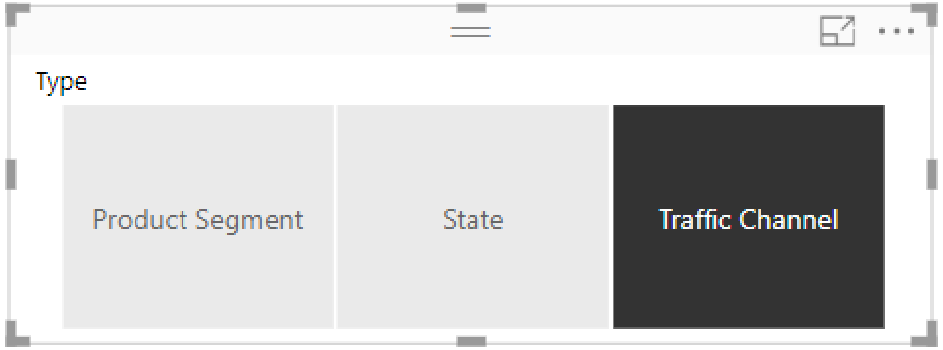


```
Dynamic Dimension =
VAR d = SELECTEDVALUE ( 'Rep Details'[TypeKEY] )
RETURN
    SWITCH (
        d;
        "P"; CALCULATE (
            [Total Sales];
            TREATAS ( VALUES ( 'Rep Details'[DimensionMember] ); ProductDim[Segment] )
        );
        "C"; CALCULATE (
            [Total Sales];
            TREATAS ( VALUES ( 'Rep Details'[DimensionMember] ); CampaignDim[TrafficChannel])
        );
        "G"; CALCULATE (
            [Total Sales];
            TREATAS ( VALUES ( 'Rep Details'[DimensionMember] ); GeoDim[State] )
        )
    )
```

# Dimension Switching – Implementation IV/

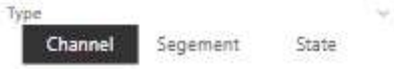


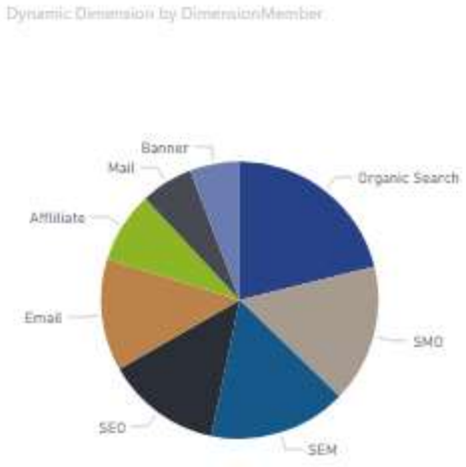
3. Create a Slicer with the column that has the type of dimension

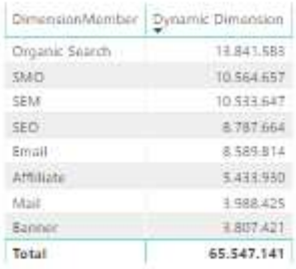


# Dimension Switching – Result

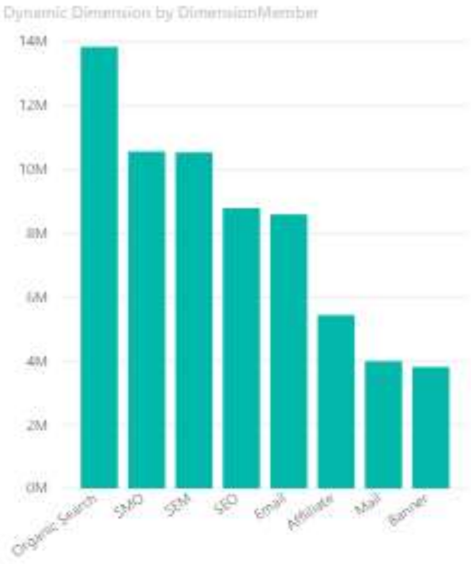
4. Then create one or more visuals to apply the measure that we created







| DimensionMember | Dynamic Dimension |
|---|---|
| Organic Search | 13.841.583 |
| SMO | 10.564.657 |
| SEM | 10.533.647 |
| SEO | 8.787.664 |
| Email | 8.589.814 |
| Affiliate | 5.433.930 |
| Mail | 3.988.425 |
| Banner | 3.807.421 |
| **Total** | **65.547.141** |



# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Module
# DAX Best Practices

# MODULE OBJECTIVES

- Emphasize importance of writing efficient DAX measures

# Use variables instead of repeating measures

- Consider the following DAX expression:

```
Ratio = IF([Total Rows] > 10, SUM(Revenue) /[Total Rows], 0)
```

- Faster DAX:

```
VAR totalRows = [Total Rows];
Ratio = IF(totalRows > 10, SUM(Revenue) / totalRows,0)
```

- In the first expression, since measures are calculated on the fly, the [Total Rows] expression gets calculated twice, first for the condition check and then for the true condition expression
- Instead of calculating the same expression multiple times, the resulting measure value can be stored in a variable and variable reference can be used wherever required

# Use DIVIDE() instead of /

- DIVIDE() function has 3rd extra parameter which is returned in case of denominator being zero

- It internally performs check to validate if the denominator is 0

- There is no need to use IF condition along with '/' operator to check for invalid denominator

- DIVIDE() also checks for ISBLANK()

- **Note:** If it is certain that the denominator value would not be 0, then it is better to use '/' operator without any *IF* check since DIVIDE() function would always perform an *IF* check internally

# Calculate ratios efficiently

**Use (a-b)/b with variables instead of a/b – 1 or a/b*100 – 100**

- We can achieve the same performance by using variables and using (a-b)/b to calculate ratio

- If both a and b are blank values, then (a-b)/b would return blank and would be filtered out where as a/b – 1 would return -1 and increase query space

# Don’t change blanks to zeros or other values

- Sometimes people replace blanks with zeros or other strings
- Power BI automatically filters out all the rows with blank values from query results
- If the blanks are replaced, the query space is greatly increased

# Use SELECTEDVALUE() instead of HASONEVALUE()

- A common pattern is to use HASONEVALUE() to check if there is only one value present for a column after applying slicers and filters and then use VALUES(ColumnName) DAX function to get the single value

- SELECTEDVALUE() performs both the above steps internally and gets the value if there is only one distinct value present for that column or returns blank in case there are multiple values available

# Use SELECTEDVALUE() instead of VALUES()

- Instead of using that, SELECTEDVALUE() must be used which is a safer function and returns blank in case of multiple values being encountered

# Use DISTINCT() and VALUES() functions consistently

- Power BI adds a Blank value to the column in case it finds referential integrity violation

- For direct query, Power BI by default adds blank value to the columns as it does not have a way to check for violations

- Difference :
  - DISTINCT(): Does not return blank which is added due to integrity violation. It includes blank only if it is part of original data
  - VALUES(): It includes blank which is added by Power BI due to referential integrity violation

- The usage of either of the function should be same throughout the whole report

- Power BI recommends to use VALUES() in the whole report if possible and blank value is not an issue

# Avoid using IFERROR() and ISERROR()

- IFERROR() and ISERROR() are sometimes used in measures
- These functions force Power BI engine to perform step by step execution of each row to check for errors as there is currently no way which directly states which row returned the error
- FIND() and SEARCH() DAX functions provide an extra parameter which can be passed and is returned in case of the search string not present – avoids use of IFERROR/ISERROR
- Both of this functions are currently also used to check for divide by zero error or along with values to check if more than one values are returned.
- Can be avoided by using the correct DAX functions like DIVIDE() and SELECTEDVALUE() which performs the error check internally and returns the expected results

# Use ISBLANK() instead of =BLANK() check

- Use inbuilt function ISBLANK() to check for any blank values instead of using comparison operator “= Blank()”

- ISBLANK() is faster

# Use FILTER(ALL(ColumnName))

- To calculate measures ignoring all the filters applied on a column, use All(ColumnName) function along with the FILTER instead of Table or VALUES().

  **E.g.: CALCULATE([Total Sales], FILTER(ALL(Products[Color]), Color = 'Red'))**

- Directly applying filters using expressions and not using FILTER function behaves in the same way as mentioned above and it internally translates to use ALL function in the filter

  **E.g.: CALCULATE([Total Sales], Products[Color]= 'Red')) ->**
  **CALCULATE([Total Sales], FILTER(ALL(Products[Color]), Products[Color]= 'Red'))**

- It is always better to apply filters at desired column than the whole table

- Always use ALL along with FILTER function if there is no specific need to keep current context

  - https://pbidax.wordpress.com/2016/05/22/simple-filter-in-dax-measures/
  - https://www.sqlbi.com/articles/filter-arguments-in-calculate/

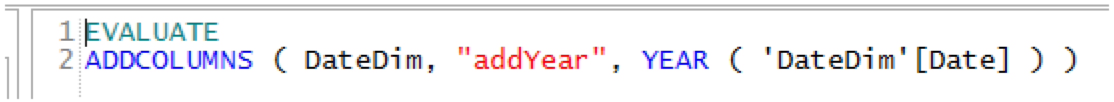
# Do not use scalar variables in SUMMARIZE()

- SUMMARIZE() traditionally used to perform grouping of columns and get the resulting aggregations along with it
- It is recommended to use SUMMARIZECOLUMNS() function which is a newer more optimized version
- SUMMARIZE function should only be used to get just the grouped elements of a table without any measures/aggregations associated with it.

  E.g. SUMMARIZE(Table, Column1, Column2)

# Avoid using ADDCOLUMNS() in measure expressions

- Measures are calculated in iterative manner by default
- If measure definitions use iterative functions like AddColumns, it create nested iteration which downgrades the performance



```
EVALUATE
ADDCOLUMNS ( DateDim, "addYear", YEAR ( 'DateDim'[Date] ) )
```

# Avoid string manipulation in measures

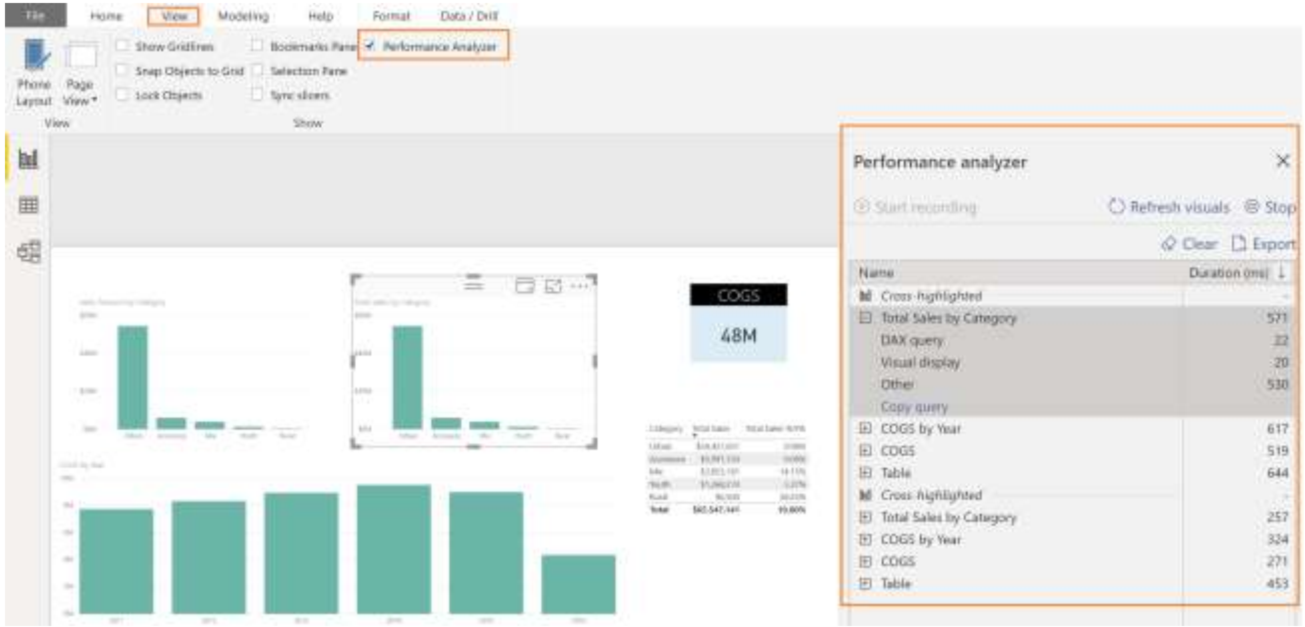
- Slows down measures
- Work is done in calculation engine

# Performance Analyzer

Using Performance Analyzer:

- You will know how each of your report elements, such as visuals and DAX formulas, are performing

- You can see and record logs that measure how each of your report elements performs when users interact with them, and which aspects of their performance are most (or least) resource intensive



| Name | Duration (ms) |
|---|---|
| Cross-highlighted | - |
| Total Sales by Category | 571 |
| DAX query | 22 |
| Visual display | 20 |
| Other | 530 |
| Copy query | |
| COGS by Year | 617 |
| COGS | 519 |
| Table | 644 |
| Cross-highlighted | - |
| Total Sales by Category | 257 |
| COGS by Year | 324 |
| COGS | 271 |
| Table | 453 |

# KNOWLEDGE CHECK Module

- Which of these are best practice ?
  - isBlank() or comparison operation =Blank()
  - SELECTEDVALUE() or HASONEVALUE()
  - DIVIDE() or IFERROR()
  - Using variables or repeating calculations

# KNOWLEDGE CHECK ANSWERS Module

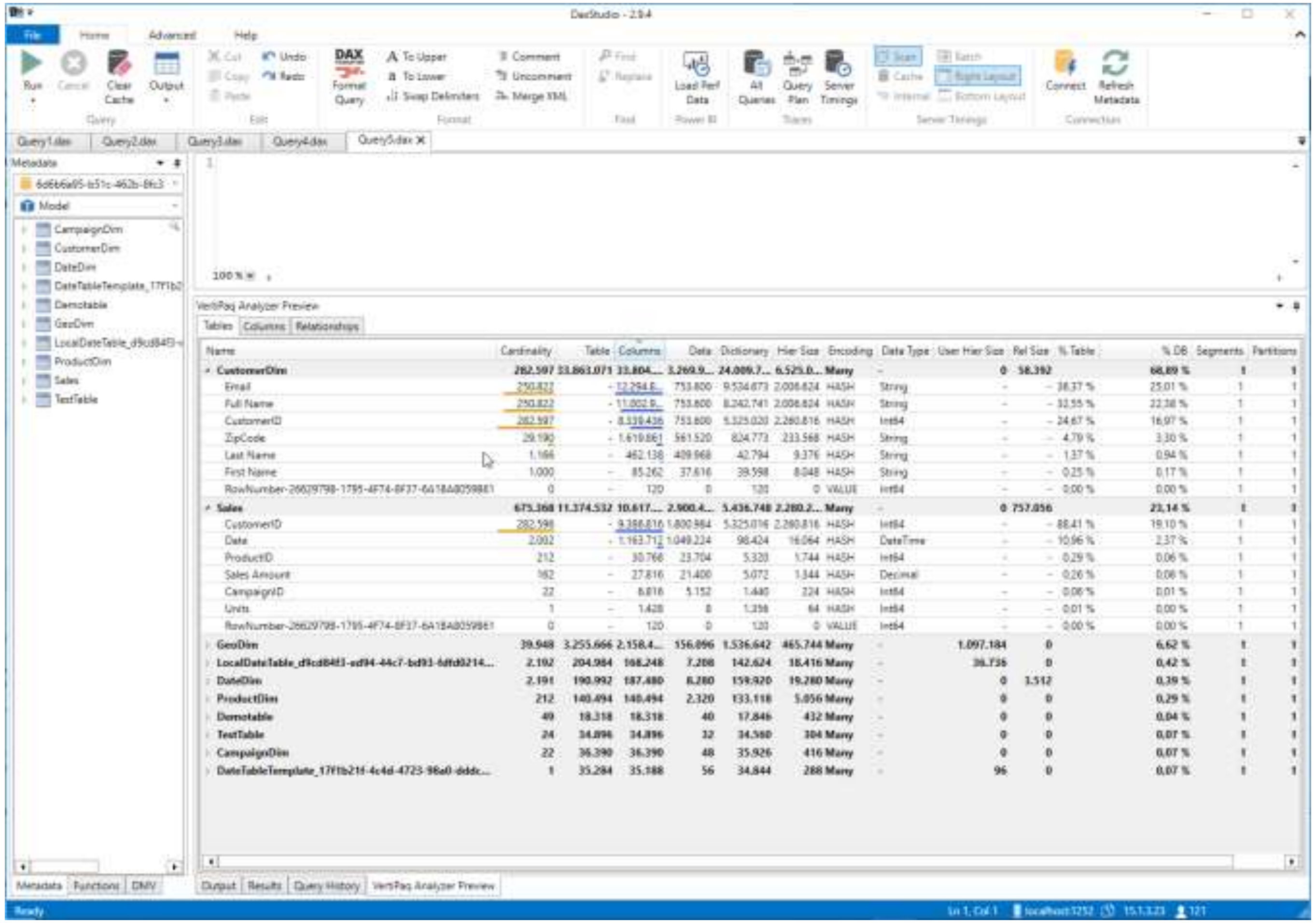
- Which of these are best practice ?
  - **isBlank()** or comparison operation =Blank()
  - **SELECTEDVALUE()** or HASONEVALUE()
  - **DIVIDE()** or IFERROR()
  - **Using variables** or repeating calculations

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Dax Studio

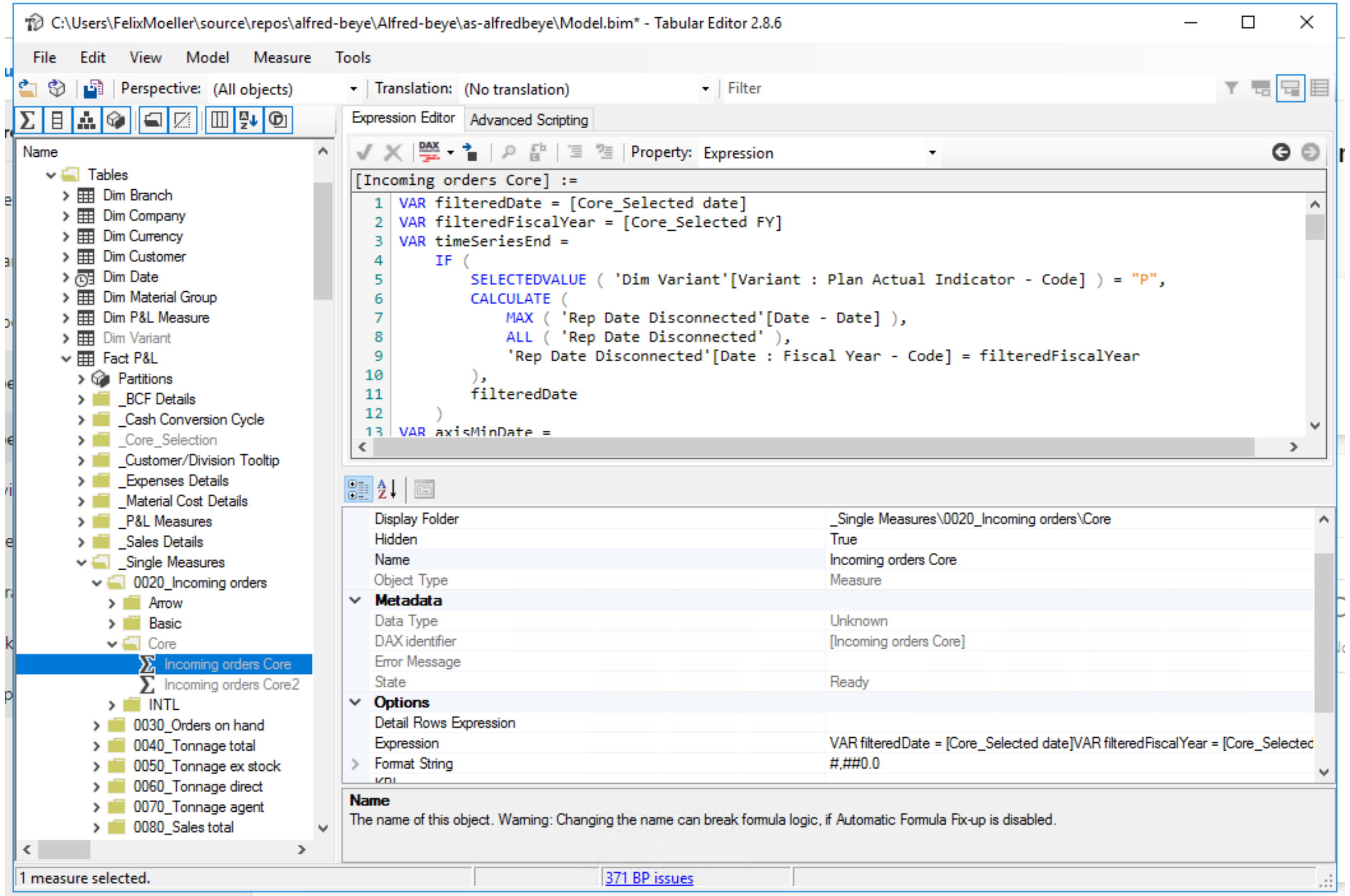
- Format Queries
- Analyze Memory Usage
- Trace Queries
- Execute Queries
- [https://github.com/DaxStudio/DaxStudio](https://github.com/DaxStudio/DaxStudio)



| Name | Cardinality | Table | Columns | Data | Dictionary | Hier Size | Encoding | Data Type | User Hier Size | Rel Size | % Table | % DB | Segments | Partitions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CustomerDim** | **282.597** | **33.863.071** | **33.804....** | **3.269.9...** | **24.009.7...** | **6.525.0...** | **Many** | - | **0** | **58.392** |  | **68,89 %** | **1** | **1** |
| Email | 250.822 | - | 12.294.6... | 753.800 | 9.534.673 | 2.008.824 | HASH | String | - | - | 38.37 % | 25.01 % | 1 | 1 |
| Full Name | 250.822 | - | 11.002.9... | 753.800 | 8.242.741 | 2.008.824 | HASH | String | - | - | 32.55 % | 22.38 % | 1 | 1 |
| CustomerID | 282.597 | - | 8.339.436 | 753.800 | 5.325.020 | 2.280.816 | HASH | Int64 | - | - | 24.67 % | 16,97 % | 1 | 1 |
| ZipCode | 29.190 | - | 1.619.861 | 561.520 | 824.773 | 233.568 | HASH | String | - | - | 4.79 % | 3.30 % | 1 | 1 |
| Last Name | 1.166 | - | 462.138 | 409.968 | 42.794 | 9.376 | HASH | String | - | - | 1.37 % | 0.94 % | 1 | 1 |
| First Name | 1.000 | - | 85.262 | 37.616 | 39.598 | 8.048 | HASH | String | - | - | 0.25 % | 0.17 % | 1 | 1 |
| RowNumber-2662979B-1795-4F74-8F37-6A1BA8059B61 | 0 | - | 120 | 0 | 120 | 0 | VALUE | Int64 | - | - | 0.00 % | 0.00 % | 1 | 1 |
| **Sales** | **675.368** | **11.374.532** | **10.617....** | **2.900.4...** | **5.436.748** | **2.280.2...** | **Many** | - | **0** | **757.056** |  | **23,14 %** | **1** | **1** |
| CustomerID | 282.598 | - | 9.388.816 | 1.800.984 | 5.325.016 | 2.280.816 | HASH | Int64 | - | - | 88.41 % | 19.10 % | 1 | 1 |
| Date | 2.002 | - | 1.163.712 | 1.049.224 | 98.424 | 16.064 | HASH | DateTime | - | - | 10.96 % | 2.37 % | 1 | 1 |
| ProductID | 212 | - | 30.766 | 23.704 | 5.320 | 1.744 | HASH | Int64 | - | - | 0.29 % | 0.06 % | 1 | 1 |
| Sales Amount | 162 | - | 27.816 | 21.400 | 5.072 | 1.344 | HASH | Decimal | - | - | 0.26 % | 0.06 % | 1 | 1 |
| CampaignID | 22 | - | 6.816 | 5.152 | 1.440 | 224 | HASH | Int64 | - | - | 0.06 % | 0.01 % | 1 | 1 |
| Units | 1 | - | 1.428 | 8 | 1.258 | 64 | HASH | Int64 | - | - | 0.01 % | 0.00 % | 1 | 1 |
| RowNumber-2662979B-1795-4F74-8F37-6A1BA8059B61 | 0 | - | 120 | 0 | 120 | 0 | VALUE | Int64 | - | - | 0.00 % | 0.00 % | 1 | 1 |
| **GeoDim** | **39.948** | **3.255.666** | **2.158.4...** | **156.096** | **1.536.642** | **465.744** | **Many** | - | **1.097.184** | **0** |  | **6,62 %** | **1** | **1** |
| **LocalDateTable_d9cd84f3-ed94-44c7-bd93-6dfd0214...** | **2.192** | **204.984** | **168.248** | **7.208** | **142.624** | **18.416** | **Many** | - | **36.736** | **0** |  | **0,42 %** | **1** | **1** |
| **DateDim** | **2.191** | **190.992** | **187.480** | **8.280** | **159.920** | **19.280** | **Many** | - | **0** | **3.512** |  | **0,39 %** | **1** | **1** |
| **ProductDim** | **212** | **140.494** | **140.494** | **2.320** | **133.118** | **5.056** | **Many** | - | **0** | **0** |  | **0,29 %** | **1** | **1** |
| **Demotable** | **49** | **18.318** | **18.318** | **40** | **17.846** | **432** | **Many** | - | **0** | **0** |  | **0,04 %** | **1** | **1** |
| **TestTable** | **24** | **34.896** | **34.896** | **32** | **34.560** | **304** | **Many** | - | **0** | **0** |  | **0,07 %** | **1** | **1** |
| **CampaignDim** | **22** | **36.390** | **36.390** | **48** | **35.926** | **416** | **Many** | - | **0** | **0** |  | **0,07 %** | **1** | **1** |
| **DateTableTemplate_17f1b21f-4c4d-4723-98a0-dddc...** | **1** | **35.284** | **35.188** | **56** | **34.844** | **288** | **Many** | - | **96** | **0** |  | **0,07 %** | **1** | **1** |

# Tabular Editor

- Stable Editor for Analysis Services Models (same engine as Power BI)
- Automation of measure creation
- [https://github.com/otykier/TabularEditor](https://github.com/otykier/TabularEditor)



```
[Incoming orders Core] :=
VAR filteredDate = [Core_Selected date]
VAR filteredFiscalYear = [Core_Selected FY]
VAR timeSeriesEnd =
    IF (
        SELECTEDVALUE ( 'Dim Variant'[Variant : Plan Actual Indicator - Code] ) = "P",
        CALCULATE (
            MAX ( 'Rep Date Disconnected'[Date - Date] ),
            ALL ( 'Rep Date Disconnected' ),
            'Rep Date Disconnected'[Date : Fiscal Year - Code] = filteredFiscalYear
        ),
        filteredDate
    )
VAR axisMinDate =
```

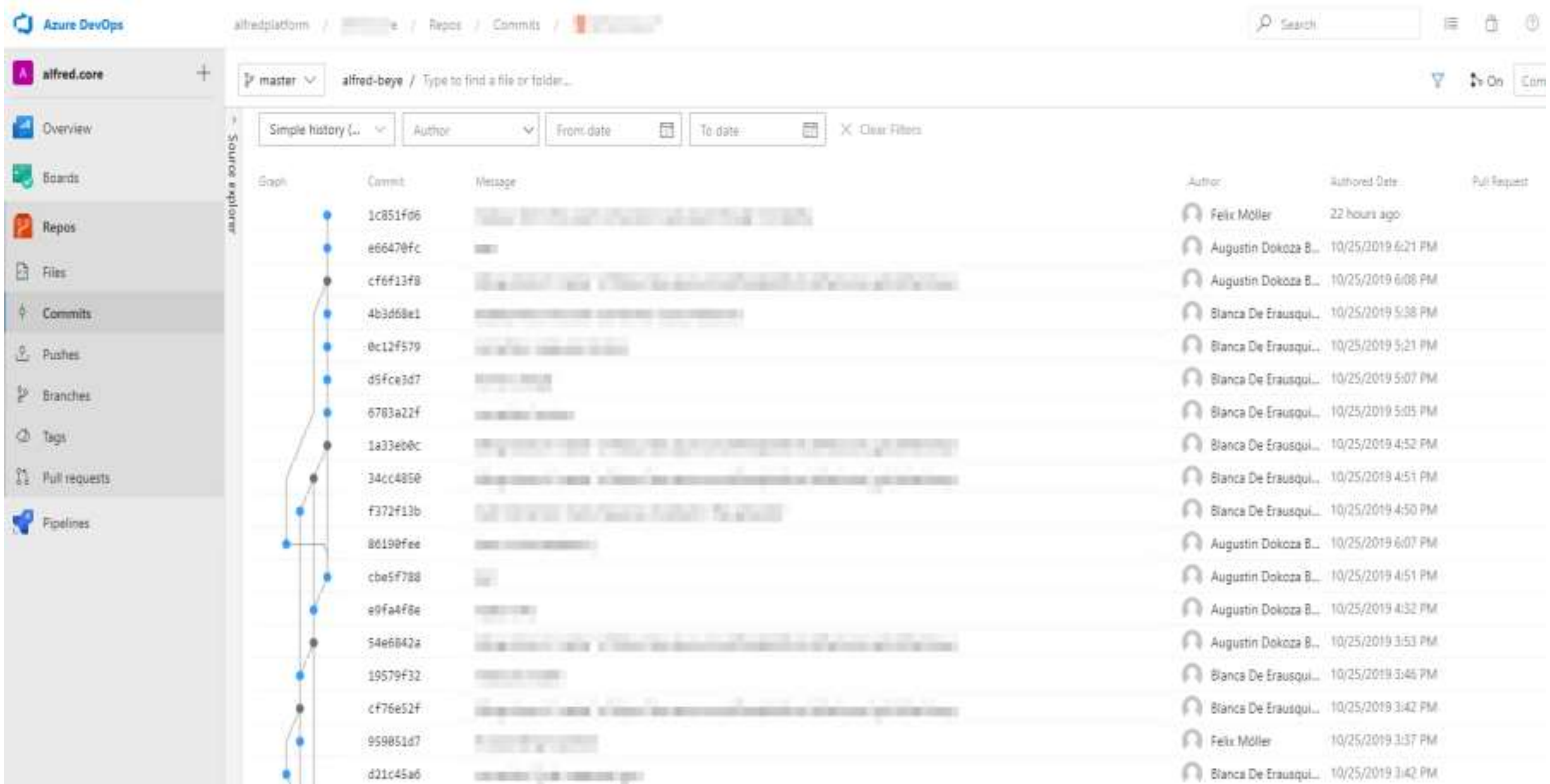
| | |
|---|---|
| Display Folder | _Single Measures\0020_Incoming orders\Core |
| Hidden | True |
| Name | Incoming orders Core |
| Object Type | Measure |
| **Metadata** | |
| Data Type | Unknown |
| DAX identifier | [Incoming orders Core] |
| Error Message | |
| State | Ready |
| **Options** | |
| Detail Rows Expression | |
| Expression | VAR filteredDate = [Core_Selected date]VAR filteredFiscalYear = [Core_Selectec |
| Format String | #,##0.0 |

**Name**
The name of this object. Warning: Changing the name can break formula logic, if Automatic Formula Fix-up is disabled.

1 measure selected.   371 BP issues

# Azure Repos

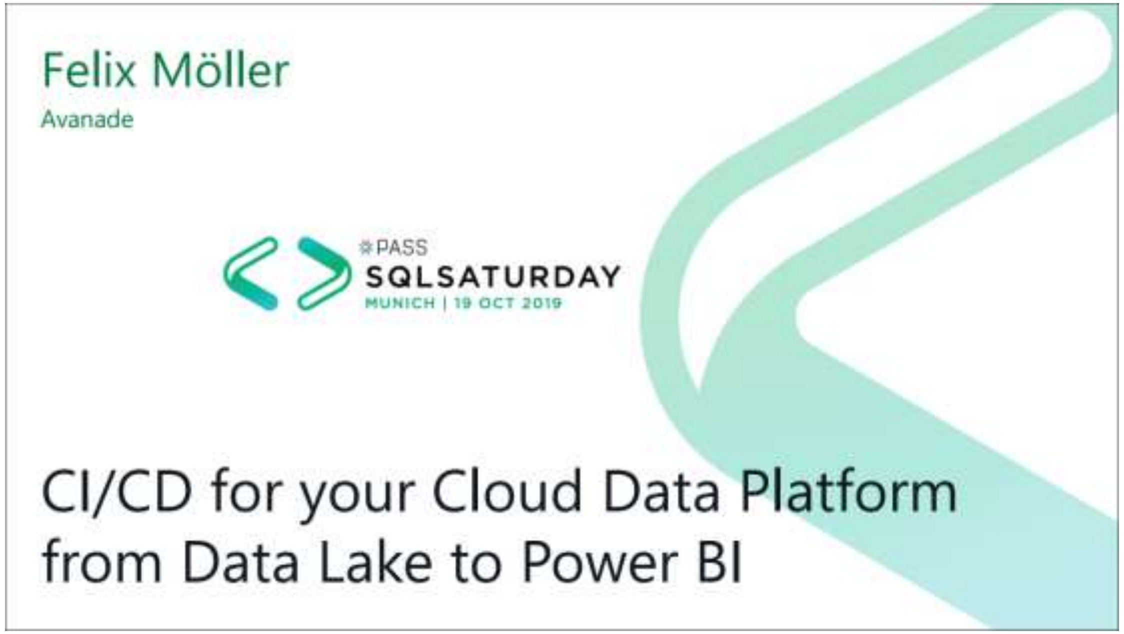
- Source control is important
- Power BI currently is very bad for source control
- [https://dev.azure.com](https://dev.azure.com)



| Graph | Commit | Message | Author | Authored Date | Pull Request |
|---|---|---|---|---|---|
| | 1c851fd6 | [redacted] | Felix Möller | 22 hours ago | |
| | e66470fc | [redacted] | Augustin Dokoza B... | 10/25/2019 6:21 PM | |
| | cf6f13f8 | [redacted] | Augustin Dokoza B... | 10/25/2019 6:08 PM | |
| | 4b3d68e1 | [redacted] | Blanca De Erausqui... | 10/25/2019 5:38 PM | |
| | 0c12f579 | [redacted] | Blanca De Erausqui... | 10/25/2019 5:21 PM | |
| | d5fce3d7 | [redacted] | Blanca De Erausqui... | 10/25/2019 5:07 PM | |
| | 6783a22f | [redacted] | Blanca De Erausqui... | 10/25/2019 5:05 PM | |
| | 1a33eb0c | [redacted] | Blanca De Erausqui... | 10/25/2019 4:52 PM | |
| | 34cc4850 | [redacted] | Blanca De Erausqui... | 10/25/2019 4:51 PM | |
| | f372f13b | [redacted] | Blanca De Erausqui... | 10/25/2019 4:50 PM | |
| | 86190fee | [redacted] | Augustin Dokoza B... | 10/25/2019 6:07 PM | |
| | cbe5f788 | [redacted] | Augustin Dokoza B... | 10/25/2019 4:51 PM | |
| | e0fa4f8e | [redacted] | Augustin Dokoza B... | 10/25/2019 4:32 PM | |
| | 54e6842a | [redacted] | Augustin Dokoza B... | 10/25/2019 3:53 PM | |
| | 19579f32 | [redacted] | Blanca De Erausqui... | 10/25/2019 3:46 PM | |
| | cf76e52f | [redacted] | Blanca De Erausqui... | 10/25/2019 3:42 PM | |
| | 959851d7 | [redacted] | Felix Möller | 10/25/2019 3:37 PM | |
| | d21c45a6 | [redacted] | Blanca De Erausqui... | 10/25/2019 3:42 PM | |

# Azure DevOps

- At Avanade we heavily use Azure DevOps to automate deployments
- See my talk at SQL Saturday two weeks ago https://www.sqlsaturday.com/880/Sessions/Details.aspx?sid=92840



Felix Möller

Avanade

PASS SQLSATURDAY MUNICH | 19 OCT 2019

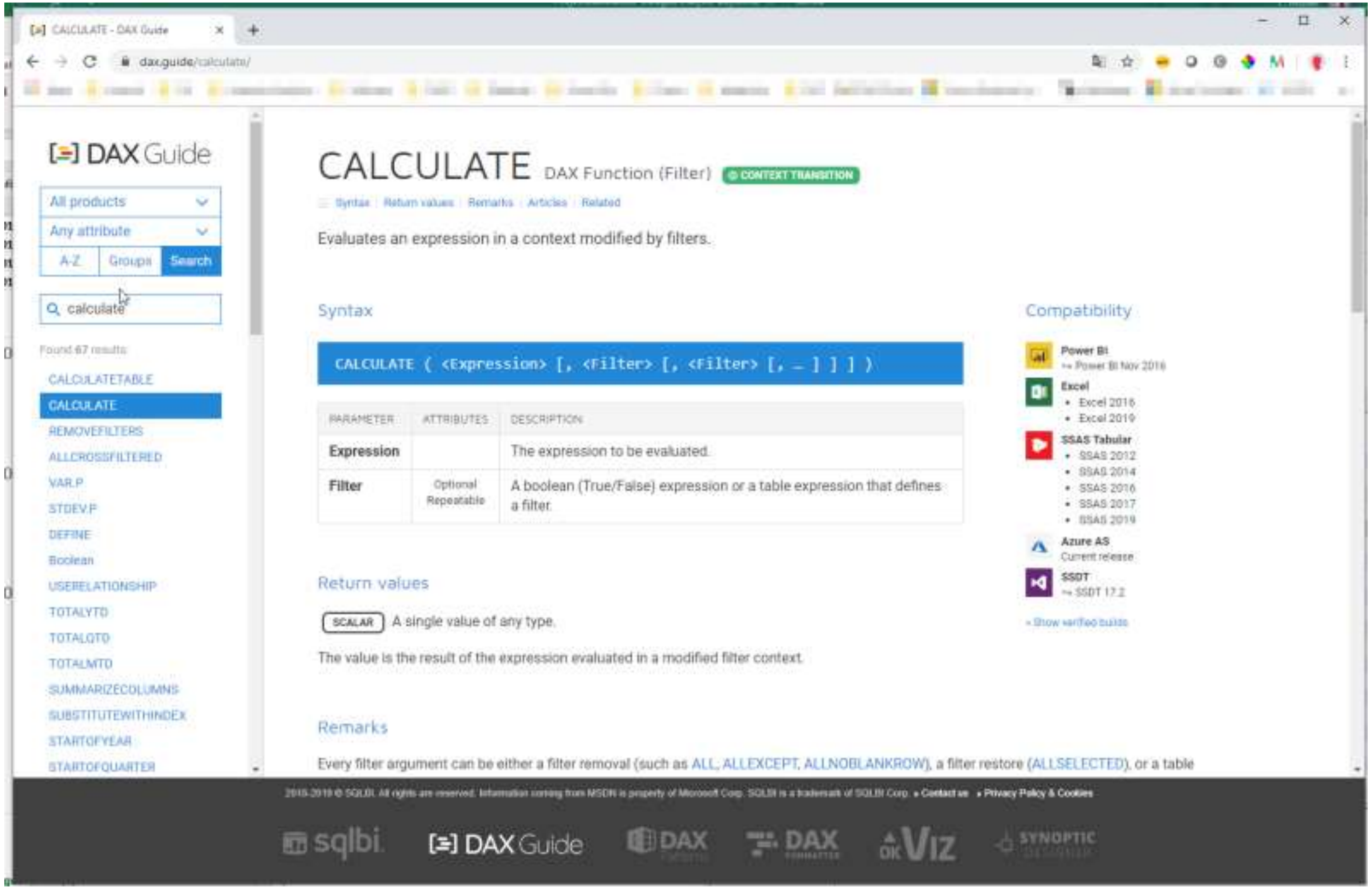
CI/CD for your Cloud Data Platform from Data Lake to Power BI

# PowerShell Power BI cmdlets

- Allow to access Power BI workspaces to be accessed from PowerShell
- https://github.com/microsoft/powerbi-powershell

# DAX Guide

- https://dax.guide/



Found 67 results

- CALCULATETABLE
- CALCULATE
- REMOVEFILTERS
- ALLCROSSFILTERED
- VAR.P
- STDEV.P
- DEFINE
- Boolean
- USERELATIONSHIP
- TOTALYTD
- TOTALQTD
- TOTALMTD
- SUMMARIZECOLUMNS
- SUBSTITUTEWITHINDEX
- STARTOFYEAR
- STARTOFQUARTER

## CALCULATE DAX Function (Filter) CONTEXT TRANSITION

Syntax | Return values | Remarks | Articles | Related

Evaluates an expression in a context modified by filters.

### Syntax

```
CALCULATE ( <Expression> [, <Filter> [, <Filter> [, … ] ] ] )
```

| PARAMETER | ATTRIBUTES | DESCRIPTION |
|---|---|---|
| **Expression** | | The expression to be evaluated. |
| **Filter** | Optional<br>Repeatable | A boolean (True/False) expression or a table expression that defines a filter. |

### Return values

SCALAR A single value of any type.

The value is the result of the expression evaluated in a modified filter context.

### Remarks

Every filter argument can be either a filter removal (such as ALL, ALLEXCEPT, ALLNOBLANKROW), a filter restore (ALLSELECTED), or a table

### Compatibility

**Power BI**
↦ Power BI Nov 2016

**Excel**
- Excel 2016
- Excel 2019

**SSAS Tabular**
- SSAS 2012
- SSAS 2014
- SSAS 2016
- SSAS 2017
- SSAS 2019

**Azure AS**
Current release

**SSDT**
↦ SSDT 17.2

» Show verified builds

2015-2019 © SQLBI. All rights are reserved. Information coming from MSDN is property of Microsoft Corp. SQLBI is a trademark of SQLBI Corp. » Contact us » Privacy Policy & Cookies

avanade

# Agenda

| | | | |
|---|---|---|---|
| 1 | **Introduction and Overview** | Together | 09:00 – 09:15 |
| 2 | **DAX Modelling Basis & Power BI Desktop Internals** | Felix M. | 09:15 – 10:15 |
| | ***Mid Morning break*** | | *10:15 – 10:30* |
| 3 | **DAX Calculated Columns & Measures** | Augustin B. | 10:30 – 11:30 |
| 4 | **CALCULATE** | Felix M. | 11:30 – 12:00 |
| | ***Lunch*** | | *12:00 – 13:00* |
| 5 | **DAX Evaluation Contexts** | Augustin B. | 13:00 – 14:00 |
| 6 | **Data Modelling: Time Intelligence Functions** | Felix M. | 14:00 – 14:30 |
| 7 | **DAX Modelling: Measure and Dimension Switching** | Felix M. | 14:30 – 15:00 |
| | ***Afternoon break*** | | *15:00 – 15:15* |
| 8 | **DAX Best Practices** | Augustin B. | 15:15 – 15:45 |
| 9 | **Essential Tools** | Felix M. | 15:45 – 16:45 |
| 10 | **Questions** | Together | 16:45 – 17:00 |

# Questions?

Any open questions? We are happy to help!

# Training Materials

The training material is updated continuously by Microsoft
https://community.powerbi.com/t5/Community-Blog/Power-BI-Training-Content/ba-p/807161#AdvModeling

avanade